AIP-5SSB0

Adaptive Information Processing

Lecture Notes

Bert de Vries
Tjalling Tjalkens (Flux 7.101)
Marco Cox (Flux 7.060)

Course Outline and Administrative Issues

Adaptive Information Processing (5SSB0)

- **When**: 3rd quartile, at 8 weeks of 4 hours per week.
- **Load**: Total workload is 5 ECTS $\Rightarrow 5 \times 28 [\text{hrs/ECTS}] = 140$ hours or $140/32 \approx 4.4$ study hours per lecture.
- **Web**: http://bertdv.github.io/teaching/AIP-5SSB0/

Course comes in Two Parts

Part I: Linear Gaussian Models and the EM Algorithm

- $4 \times 4 = 16$ lecture hours, mostly in February
- Instructor: Bert de Vries, rm. FLUX-7.060 (on Wednesdays)
- email bdevries@gnresound.com
- Code examples in Julia
  - Consult the **teaching assistant** Marco Cox (FLUX-7.060; email mail@marcocox.com) for help on Julia.

Part II: Model Complexity Control and the MDL Principle

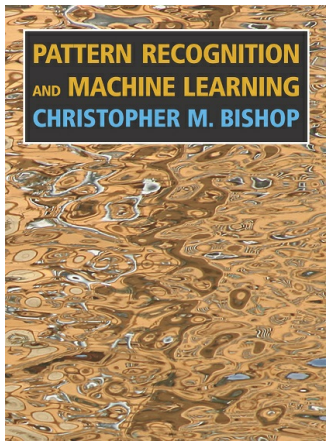
- $4 \times 4 = 16$ lecture hours, mostly in March
- Instructor: Tjalling Tjalkens, rm. FLUX-7.101
- email

Why Take This Class?

- Suppose you need to develop an algorithm for a complex DSP task. This is what you'll do:
  1. Choose a set of candidate algorithms $y = H_k(x;\theta)$ where $k \in \{1,2,\ldots,K\}$ and $\theta \in \Theta_k$; (you think that) there's an algorithm $H_{k^*}(x;\theta^*)$ that performs according to your liking.
  2. You collect a set of examples $D = \{(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)\}$ that are consistent with the correct algorithm behavior.
- Using the methods from this class, you will be able to do:
  1. **model selection**, i.e., find k^*
  2. **parameter estimation**, i.e., find θ^*

Materials

- Book:



- find at http://research.microsoft.com/~cmbishop/PRML/index.htm.
- Background reading; covers about the same stuff as (mandatory) slides.
- Contains much more material; great for future study and reference.

Exam Guide Part–1 (LGM + EM)

- Tested material consists of these lecture notes, reading assignments (see website) and exercises.
- Slides that are not required for the exam are indicated by the word (OPTIONAL) in the header.
- Book (Bishop) readings optional unless indicated through exercises.
- Advice: download (and make free use of) Sam Roweis' cheat sheets for Matrix and Gaussian formulas.
- **Very strong advice** Make old exams!
- Further exam instructions for part-2 from Tjalling.

Machine Learning Overview

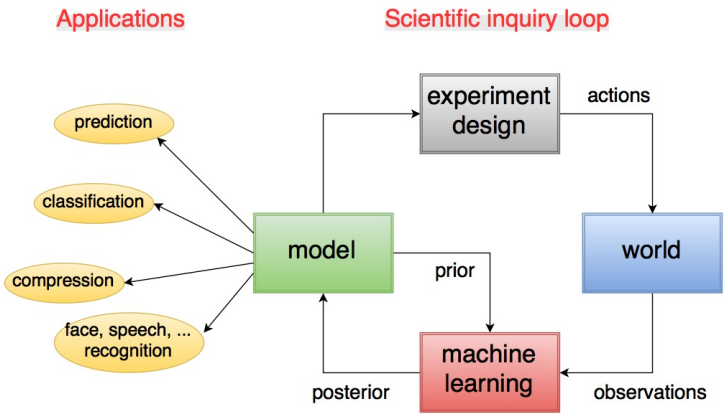
Preliminaries

- Goal
  - Top-level overview of machine learning
- Materials
  - Study Bishop pp. 1-4
  - Study this notebook

What is Machine Learning?

- Machine Learning relates to **building models from data**.
  - Suppose we want to make a model for a complex process about which we have little knowledge (so hand-programming is not possible).
  - **Solution**: Get the computer to program itself by showing it examples of the behavior that we want.
  - Practically, we choose a library of models, and write a program that picks a model and tunes it to fit the data.
  - **Criterion**: a *good* model generalizes well to unseen data from the same process.
- This method is known in various scientific communities under different names such as machine learning, statistical inference, system identification, data mining, source coding, data compression, etc.

Machine learning and the scientific inquiry loop.



Machine Learning is Difficult

- Modeling (Learning) Problems
  - Is there any regularity in the data anyway?
  - What is our prior knowledge and how to express it mathematically?
  - How to pick the model library?
  - How to tune the models to the data?
  - How to measure the generalization performance?
- Quality of Observed Data
  - Not enough data
  - Too much data?
  - Available data may be messy (measurement noise, missing data points, outliers)

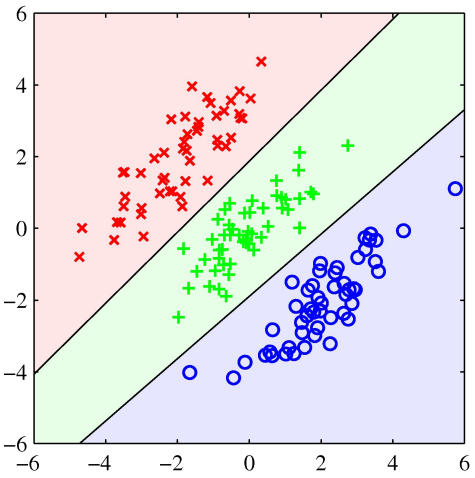
A Machine Learning Taxonomy

1. **Supervised Learning**: Given examples of inputs and corresponding desired outputs, predict outputs on future inputs.
   - Examples: classification, regression, time series prediction
2. **Unsupervised Learning**: (a.k.a. **density estimation**). Given only inputs, automatically discover representations, features, structure, etc.
   - Examples: clustering, outlier detection, compression
3. **Reinforcement Learning**: Given sequences of inputs, actions from a fixed set, and scalar rewards/punishments, *learn* to select action sequences in a way that maximizes expected reward, e.g. chess and robotics. (This is more akin to learning how to design good experiments and is not covered in this course.)
4. Other stuff, like **Preference Learning**, **learning to rank**, etc. (also not covered in this course)

Supervised Learning

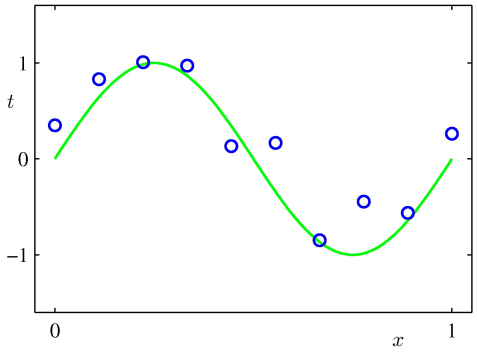
- Given observations $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, the goal is to estimate the conditional distribution $p(y|x)$.

Classification



- The target variable y is a *discrete-valued* vector representing class labels

Regression

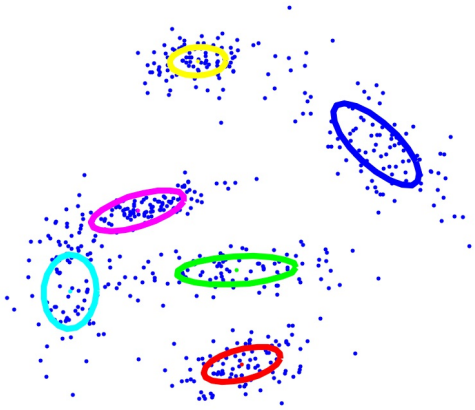


- Same problem statement as classification but now the target variable is a *real-valued* vector.

Unsupervised Learning

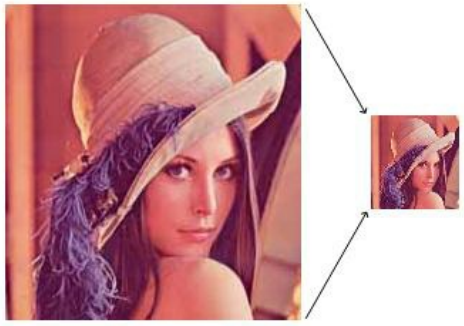
Given data $D = \{x_1, \ldots, x_N\}$, model the (unconditional) probability distribution $p(x)$ (a.k.a. **density estimation**).

Clustering



- Group data into clusters such that all data points in a cluster have similar properties.

Compression / dimensionality reduction



- Output from coder is much smaller in size than original, but if coded signal if further processed by decoder, then the result is very close (or exactly equal) to the original.

Some Machine Learning Applications

- computer speech recognition, speaker recognition
- face recognition, iris identification
- printed and handwritten text parsing
- financial prediction, outlier detection (credit-card fraud)
- user preference modeling (amazon); modeling of human perception
- modeling of the web (google)
- machine translation
- medical expert systems for disease diagnosis (e.g., mammogram)
- strategic games (chess, go, backgammon)
- **any 'knowledge-poor' but 'data-rich' problem**

Probability Theory Review

Preliminaries

- Goal
  - Review of probability theory from a logical viewpoint (i.e., a Bayesian interpretation)
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 12-20
    - Bruininkx - 2002 - Bayesian Probability

Why Probability Theory?

- Probability theory (PT) is the **theory of optimal processing of incomplete information**, and as such provides a quantitative framework for drawing conclusions from a finite (read: incomplete) data set.
- Machine learning concerns drawing conclusions from data.
- In general, nearly all interesting questions in machine learning can be stated in the following form (a conditional probability):
$$p(\text{whatever-we-want-to-know} \mid \text{whatever-we-do-know})$$
- For example:
  - Predictions
$$p(\text{ future-observations} \mid \text{past-observations})$$
  - Classify a received data point
$$p(\text{ } x\text{-belongs-to-class-}k \mid x \text{ })$$
  - Compress data in an efficient way
$$p(\text{ } x_n \mid x_{1:n-1} \text{ })$$
- Note that **Information theory** (the "theory of log-probability") provides a source coding view on machine learning that is consistent with probability theory (more in part-2).

Probability Theory Notation

- An **event** A is a statement, whose truth is contemplated by a person, e.g.,
$$A = \text{'it will rain tomorrow'}$$
- The denial of event A, **not**-A, is written as $\bar{A}$.
- For any event A, with background knowledge I, the **conditional probability of A, given I** is written as
$$p(A|I)\,,$$
which is a number between 0 and 1. If, given I, you know that A is true, than $p(A|I) = 1$.
- The probability $p(A|I)$ should be **interpreted as your degree of belief** that event A is true, given that I is true. This is an extremely powerful interpretation (later more about this).

Notation for **compound events**:

- The **joint** probability that both A and B are true, given I (a.k.a. **conjunction**) is written as
$$p(A, B|I)$$
- The probability that either A or B, or both A and B, are true, given I (a.k.a. **disjunction**) is written as
$$p(A + B|I)$$

Probability Theory Calculus

Probability theory **extends** boolean logic to rational reasoning with uncertainty. Under some mild conditions, the following calculation rules can be derived, (see Cox, 1946):

- **Product rule**. For 2 events A, B with given background I,
$$p(A,B|I) = p(A|B,I)\,p(B|I)\,.$$
- **Sum rule**. For 2 events A, B with given background I,
$$p(A+B|I) = p(A|I) + p(B|I) - p(A,B|I)\,.$$
- Clearly, it follows from the sum rule that $p(A|I) + p(\bar{A}|I) = 1$
- Note that the background information may not change, e.g., if $I' \neq I$, then
$$p(A,B|I) \neq p(A|B,I)\,p(B|I')$$
- **All legitimate relations between probabilities can be derived from the sum and product rules!!**

Some Notation and useful facts

- Iff
$$p(A,B|I) = p(A|I)p(B|I)$$
then A and B are said to be **independent**, given I. Note that this is equivalent to the statement $p(A|B,I) = p(A|I)$.
- The product and sum rules are also known as the **axioms of probability theory**, but in fact they can be derived as the unique rules for rational reasoning under uncertainty (Cox, 1946).
- All probabilities are in principle conditional probabilities of the type $p(A|I)$, since there is always some background knowledge.
- Shorthand notation
  1. We often write $p(A)$ rather than $p(A|I)$ if the background knowledge I is assumed to be obviously present. E.g., $p(X=5)$ rather than $p(\,X=5 \mid \text{the-sun-comes-up-tomorrow}\,)$.
  2. If X is a random variable and $X=x$ is an event, then we often write $p(x)$ rather than $p(X=x)$ (hoping again that the reader understands the context ;-)
  3. $p(X)$ denotes the distribution over random variable X.
- Next, we present two useful corollaries: (1) Marginalization and (2) Bayes rule

The Sum Rule and Marginalization

- If X and Y are random variables over finite domains, than it follows from the sum rule that
$$p(X) = \sum_Y p(X,Y) = \sum_Y p(X|Y)p(Y)\,.$$
- Note that this is just a *generalized sum rule*. In fact, Bishop (p.14) (and some other authors as well) calls this the sum rule (which is not wrong, it's just a rewrite).
  - EXERCISE: Proof the generalized sum rule.
- Of course, in the continuous domain, the (generalized) sum rule becomes
$$p(X) = \int p(X,Y)\,\mathrm{d}Y$$
- Integrating Y out of a joint distribution is called **marginalization** and the result $p(X)$ is sometimes referred to as the **marginal probability**.

The Product Rule and Bayes Rule

- Consider 2 variables D and θ; it follows symmetry arguments that
$$p(D,\theta) = p(D|\theta)p(\theta) = p(\theta|D)p(D)$$
and hence that
$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}\,.$$

- This formula is called **Bayes rule** (or Bayes theorem). While Bayes rule is always true, a particularly useful application occurs when D refers to an observed data set and θ is set of model parameters that relates to the data. In that case,
  - the **prior** probability $p(\theta)$ represents our **degree-of-belief** about proper values for θ, before seeing the data D.
  - the **posterior** probability $p(\theta|D)$ relates to our state-of-knowledge about θ after we have seen the data.

$\Rightarrow$ Bayes rule tells us how to update our knowledge about model parameters (or other hypotheses) when facing new data. Hence,

> Bayes rule is the fundamental rule for machine learning!

Bayes Rule Nomenclature

- Some nomenclature associated with Bayes rule:
$$\underbrace{p(\theta|D)}_{\text{posterior}} = \frac{\overbrace{p(D|\theta)}^{\text{likelihood}} \times \overbrace{p(\theta)}^{\text{prior}}}{\underbrace{p(D)}_{\text{evidence}}}$$

- Note that the evidence can be computed from the numerator through marginalization since
$$p(D) = \int p(D,\theta)\,\mathrm{d}\theta = \int p(D|\theta)\,p(\theta)\,\mathrm{d}\theta$$

- Hence, likelihood and prior is sufficient to compute both the evidence and the posterior. To emphasize that point, Bayes rule is sometimes written as
$$p(\theta|D)\,p(D) = p(D|\theta)\,p(\theta)$$

- For given D, the posterior probabilities of the parameters scale relatively against each other as
$$p(\theta|D) \propto p(D|\theta)p(\theta)$$

$\Longrightarrow$ All that we can learn from the observed data is contained in the likelihood function $p(D|\theta)$. This is called the **likelihood principle**.

The Likelihood Function

Consider a probabilistic model $p(D|\theta)$, where D relates to a data set and θ are model parameters.

- In general, $p(D|\theta)$ is a function of both D and θ.
- The **sampling distribution**
$$p(D|\theta = \theta_0)$$
describes the probability that data D is observed, assuming that it is generated by the given model with parameter values set to $\theta = \theta_0$.
- In a machine learning context, often $D = D_0$ is given (observed), and θ is the free variable.
- When viewed as a function of the free variable θ with given $D = D_0$,
$$\mathrm{L}(\theta) \triangleq p(D = D_0|\theta)$$
is called the **likelihood function** (for θ)
- Note that $\mathrm{L}(\theta)$ is not a probability distribution for θ. (Is $\sum_\theta \mathrm{L}(\theta) = 1$ always true?)

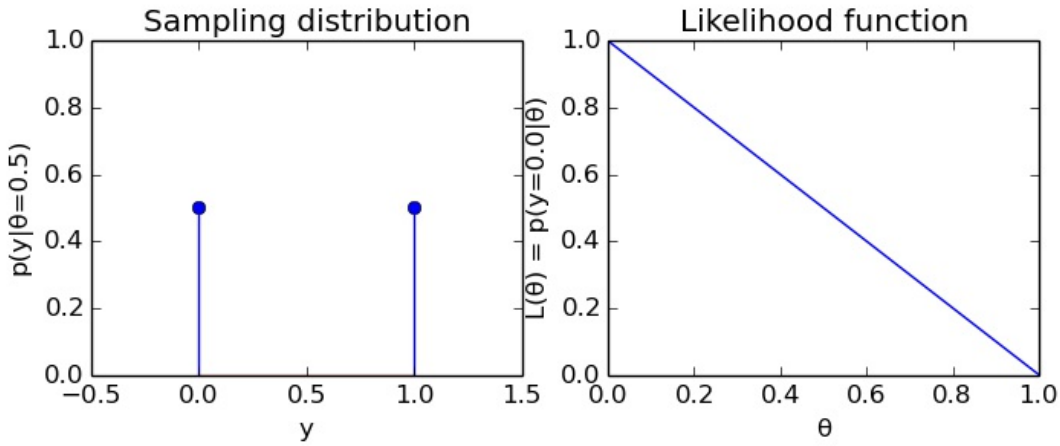
CODE EXAMPLE

Consider the following simple model for the outcome of the tossing of a biased coin with parameter $\theta \in [0,1]$:

$$y \in \{0,1\}$$
$$p(y|\theta) \triangleq \theta^y (1-\theta)^{1-y}$$
$$p(D|\theta) = p(y_1, \ldots, y_N|\theta) = \prod_{n=1}^{N} p(y_n|\theta)$$

If we consider just one data point, we can plot both the sampling distribution (i.e. $p(y|\theta = 0.8)$) and the likelihood function (i.e. $L(\theta) = p(y = 0|\theta)$).

```
using Reactive, Interact, PyPlot
p(y,θ) = θ.^y .* (1-θ).^(1-y)
f = figure()
@manipulate for y=false, θ=0:0.1:1; withfig(f) do
        # Plot the sampling distribution
        subplot(221); stem([0,1], p([0,1],θ));
        title("Sampling distribution");
        xlim([-0.5,1.5]); ylim([0,1]); xlabel("y"); ylabel("p(y|θ=$(θ))");
        # Plot the likelihood function
        _θ = linspace(0.0, 1.0, 100)
        subplot(222); plot(_θ, p(convert(Float64,y), _θ));
        title("Likelihood function");
        xlabel("θ");
        ylabel("L(θ) = p(y=$(convert(Float64,y))|θ)");
    end
end
```



The (discrete) sampling distribution is a valid probability distribution. However, the likelihood function $L(\theta)$ clearly isn't, since $\int_0^1 L(\theta) \mathrm{d}\theta \neq 1$.

END OF CODE EXAMPLE

Probabilistic Inference

- **Probabilistic inference** refers to computing

$$p(\text{ whatever-we-want-to-know} \mid \text{whatever-we-already-know})$$

- For example:

$$\begin{aligned} p(\text{ Mr.S.-killed-Mrs.S.} &\mid \text{he-has-her-blood-on-his-shirt}) \\ p(\text{ transmitted-codeword} &\mid \text{received-codeword}) \\ p(\text{ articulatory-movements} &\mid \text{speech-signal}) \\ p(\text{ fetal-HR-signal} &\mid \text{mother-EMG-signal}) \end{aligned}$$

- This can be accomplished by repeated application of sum and product rules, (of course).

- For instance, consider a joint distribution $p(X,Y,Z)$. Assume we are interested in $p(X|Z)$:

$$\begin{aligned} p(X|Z) &\overset{p}{=} \frac{p(X,Z)}{p(Z)} \\ &\overset{s}{=} \frac{\int_Y p(X,Y,Z)}{\int_{X,Y} p(X,Y,Z)}, \end{aligned}$$

  where the 's' and 'p' above the equality sign indicate whether the sum or product rule was used.

- In the rest of this course, we'll encounter many long probabilistic derivations. For each manipulation, you should be able to associate an 's' (for sum rule), a 'p' (for product or Bayes rule) or an 'a' (for model assumption) above any equality sign. If you can't do that, file a github issue :)

- The resulting expressions often contain a bunch of (hard) integrals and/or sums (with many terms).

Inference Exercise: Disease Diagnosis

[Question] - Given a disease D with prevalence of 1% and a test procedure T with sensitivity ('true positive' rate) of 95% and specificity ('true negative' rate) of 85%, what is the chance that somebody who tests positive actually has the disease?

[Answer] - The given data are $p(D=1) = 0.01$, $p(T=1|D=1) = 0.95$ and $p(T=0|D=0) = 0.85$. Then according to Bayes rule,

$$\begin{aligned} &p(D=1|T=1) \\ &\overset{Bayes}{=} \frac{p(T=1|D=1)p(D=1)}{p(T=1)} \\ &\overset{marg.}{=} \frac{p(T=1|D=1)p(D=1)}{p(T=1|D=1)p(D=1) + p(T=1|D=0)p(D=0)} \\ &= \frac{0.95 \times 0.01}{0.95 \times 0.01 + 0.15 \times 0.99} = 0.0601 \end{aligned}$$

Inference Exercise: Bag Counter

[Question] - A bag contains one ball, known to be either white or black. A white ball is put in, the bag is shaken, and a ball is drawn out, which proves to be white. What is now the chance of drawing a white ball?

[Answer] - Again, use Bayes and marginalization to arrive at $p(\text{white}|\text{data}) = 2/3$, see homework exercise

$\Rightarrow$ Note that probabilities describe **a person's state of knowledge** rather than a 'property of nature'.

[Excercise] - Is a speech signal a 'probabilistic' (random) or a deterministic signal?

Inference Exercise: Causality?

[Question] - A dark bag contains five red balls and seven green ones. (a) What is the probability of drawing a red ball on the first draw? Balls are not returned to the bag after each draw. (b) If you know that on the second draw the ball was a green one, what is now the probability of drawing a red ball on the first draw?

[Answer] - (a) $5/12$. (b) $5/11$, see homework.

$\Rightarrow$ Again, we conclude that conditional probabilities reflect **implications for a state of knowledge** rather than temporal causality.

PDF for the Sum of Two Variables

[Question] - Given two random **independent** variables X and Y, with PDF's $p_x(x)$ and $p_y(y)$. What is the PDF of

$$Z = X + Y \ ?$$

[Answer] - Let $p_z(z)$ be the probability that Z has value z. This occurs if X has some value x and at the same time $Y = z - x$, with joint probability $p_x(x)p_y(z-x)$. Since x can be any value, we sum over all possible values for x to get

$$p_z(z) = \int_{-\infty}^{\infty} p_x(x)p_y(z-x)\,\mathrm{d}x$$

i.e., the **convolution** of p_x and p_y.

Note that $p_z(z) \neq p_x(x) + p_y(y)$!!

$\Rightarrow$ In linear stochastic systems theory, the Fourier Transform of a PDF (i.e., the characteristic function) plays an important computational role.

CODE EXAMPLE

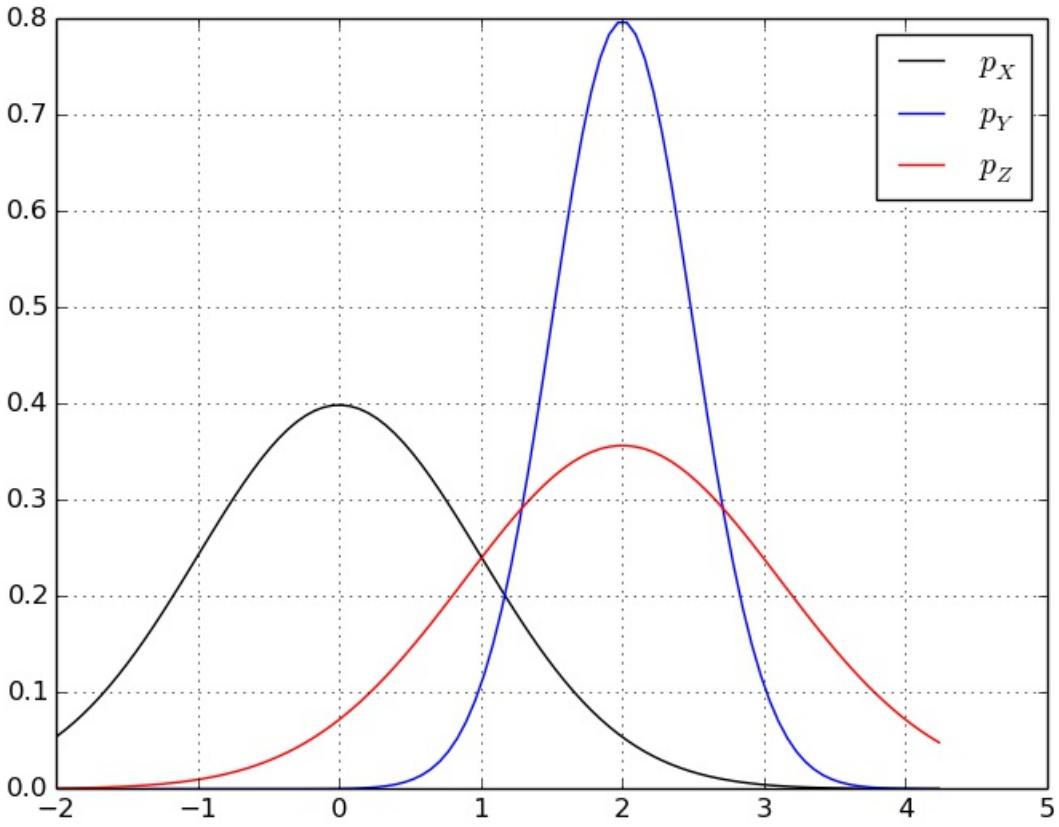
Consider the PDF of the sum of two independent Gaussians X and Y:

$$p_X(x) = \mathcal{N}(x \mid \mu_X, \sigma_X^2)$$
$$p_Y(y) = \mathcal{N}(y \mid \mu_Y, \sigma_Y^2)$$
$$Z = X + Y$$

Performing the convolution (nice exercise) yields a Gaussian PDF for Z:

$$p_Z(z) = \mathcal{N}(z \mid \mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2).$$

```
using Reactive, Interact, PyPlot, Distributions
f = figure()
@manipulate for μx=-2:0.1:2, σx=0.1:0.1:1.9,μy=0:0.1:4, σy=0.1:0.1:0.9; withfig(f) do
        μz = μx+μy; σz = sqrt(σx^2 + σy^2)
        x = Normal(μx, σx)
        y = Normal(μy, σy)
        z = Normal(μz, σz)
        range_min = minimum([μx-2*σx, μy-2*σy, μz-2*σz])
        range_max = maximum([μx+2*σx, μy+2*σy, μz+2*σz])
        range = linspace(range_min, range_max, 100)
        plot(range, pdf(x,range), "k-")
        plot(range, pdf(y,range), "b-")
        plot(range, pdf(z,range), "r-")
        legend([L"p_X", L"p_Y", L"p_Z"])
        grid()
    end
end
```



END OF CODE EXAMPLE

Expectation and Variance

- The **expected value** or **mean** is defined as

$$\mathrm{E}[f] \triangleq \int f(x)\, p(x)\, \mathrm{d}x$$

- The **variance** is defined as

$$\mathrm{var}[f] \triangleq \mathrm{E}\left[(f(x) - \mathrm{E}[f(x)])^2\right]$$

- The **covariance** matrix between *vectors* x and y is defined as

$$\begin{aligned}\mathrm{cov}[x, y] &\triangleq \mathrm{E}\left[(x - \mathrm{E}[x])(y^T - \mathrm{E}[y^T])\right] \\ &= \mathrm{E}[xy^T] - \mathrm{E}[x]\mathrm{E}[y^T]\end{aligned}$$

  - Also useful as: $\mathrm{E}[xy^T] = \mathrm{cov}[x, y] + \mathrm{E}[x]\mathrm{E}[y^T]$

Linear Transformations

No matter how x is distributed, we can easily derive that **(do as exercise)**

$$\mathrm{E}[Ax + b] = A\mathrm{E}[x] + b \qquad \text{(SRG-3a)}$$
$$\mathrm{cov}[Ax + b] = A\,\mathrm{cov}[x]\,A^T \qquad \text{(SRG-3b)}$$

- (The tag (SRG-3a) refers to the corresponding eqn number in Sam Roweis' Gaussian Identities notes.)

Example: Mean and Variance for the Sum of Two Variables

For any distribution of x and y and $z = x + y$,

$$\begin{aligned}\mathrm{E}[z] &= \int_z z \left[\int_x p_x(x) p_y(z - x)\, \mathrm{d}x\right] \mathrm{d}z \\ &= \int_x p_x(x) \left[\int_z z p_y(z - x)\, \mathrm{d}z\right] \mathrm{d}x \\ &= \int_x p_x(x) \left[\int_{y'} (y' + x) p_y(y')\, \mathrm{d}y'\right] \mathrm{d}x \\ &= \int_x p_x(x) \left(\mathrm{E}[y] + x\right) \mathrm{d}x \\ &= \mathrm{E}[x] + \mathrm{E}[y] \qquad \text{(always; follows from SRG-3a)}\end{aligned}$$

Derive as an exercise that

$$\begin{aligned}\mathrm{var}[z] &= \mathrm{var}[x] + \mathrm{var}[y] + 2\mathrm{cov}[x, y] \qquad \text{(always, see SRG-3b)} \\ &= \mathrm{var}[x] + \mathrm{var}[y] \qquad \text{(if X and Y are independent)}\end{aligned}$$

Review Probability Theory

- Interpretation as a degree of belief, i.e. a state-of-knowledge, not as a property of nature.
- We can do everything with only the **sum rule** and the **product rule**. In practice, **Bayes rule** and **marginalization** are often very useful for computing

$$p(\,\text{what-we-want-to-know} \mid \text{what-we-already-know}\,)\,.$$

- Bayes rule

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}$$

  is the fundamental rule for learning!

- That's really about all you need to know about probability theory, but you need to *really* know it, so do the exercises.

Bayesian Machine Learning

Preliminaries

- Goals
  - Introduction to probabilistic modeling
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 21-24

The Bayesian Machine Learning Framework

- Suppose that your task is to predict a `new' datum x, based on N observations $D = \{x_1, \ldots, x_N\}$.
- The Bayesian approach for this task involves three stages:
  1. Model specification
  2. parameter estimation (inference, learning)
  3. Prediction (apply the model)

Let's discuss these three stages in a bit more detail:

1. **Model specification**. Your first task is to propose a model with tuning parameters θ for generating the data D.
   - This involves specification of $p(D|\theta)$ and a prior for the parameters $p(\theta)$.
   - *You* choose the distribution $p(D|\theta)$ based on your physical understanding of the data generating process.
     - Note that, for independent observations x_n,
       $$p(D|\theta) = \prod_{n=1}^{N} p(x_n|\theta)$$
       so usually you select a model for generating one observation x_n and then use (in-)dependence assumptions to combine these models into a model for D.
   - *You* choose the prior $p(\theta)$ to reflect what you know about the parameter values before you see the data D.
2. **Parameter estimation**. After model specification, you need to measure/collect a data set D. Then, use Bayes rule to find the posterior distribution for the parameters,
   $$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} \propto p(D|\theta)p(\theta)$$
   - Note that there's **no need for you to design a *smart* parameter estimation algorithm**. The only complexity lies in the computational issues.
   - This "recipe" works only if the RHS factors can be evaluated; this is what machine learning is about
     $\Rightarrow$ **Machine learning is easy, apart from computational details:)**
3. **Prediction**. Given the data D, our knowledge about the yet unobserved datum x is captured by
   $$\begin{aligned} p(x|D) &\overset{s}{=} \int p(x,\theta|D)\,\mathrm{d}\theta \\ &\overset{p}{=} \int p(x|\theta,D)p(\theta|D)\,\mathrm{d}\theta \\ &\overset{a}{=} \int p(x|\theta)p(\theta|D)\,\mathrm{d}\theta \end{aligned}$$
   - (The symbols s, p and a relate to 'sum rule', 'product rule' and 'model assumption'. As discussed, each probabilistic manipulation will be based on the sum or product rule, or a model assumption that *you* (the designer) makes.)
   - Again, **no need to invent a special prediction algorithm**. Probability theory takes care of all that. The complexity of prediction is just computational: how to carry out the marginalization over θ.
   - In order to execute prediction, you need to have access to the factors $p(x|\theta)$ and $p(\theta|D)$. Where do these factors come from? Are they available?
   - What did we learn from D? Without access to D, we would predict new observations through
     $$p(x) = \int p(x,\theta)\,\mathrm{d}\theta = \int p(x|\theta)p(\theta)\,\mathrm{d}\theta$$
   - NB The application of the learned posterior $p(\theta|D)$ not necessarily has to be prediction. We use it here as an example, but other applications are of course also possible.

Bayesian Model Selection (OPTIONAL SLIDE)

There appears to be a remaining problem: How good really were our model assumptions $p(x|\theta)$ and $p(\theta)$?

- Technically, this is a **model selection** problem
- [Q.] What if I have more candidate models, say $\mathcal{M} = \{m_1, \ldots, m_K\}$ where each model relates to specific prior $p(\theta|m_k)$ and likelihood $p(D|\theta, m_k)$?
- [A.]: Specify a prior $p(m_k)$ for each of the models and **use Bayes again** to absorb what we can learn from the data,

$$\begin{aligned} p(m_k|D) &= \frac{p(D|m_k)p(m_k)}{p(D)} \\ &\propto p(m_k) \cdot p(D|m_k) \\ &= p(m_k) \cdot \int_\theta p(D, \theta|m_k)\, \mathrm{d}\theta \\ &= \underbrace{p(m_k)}_{\substack{\text{model} \\ \text{prior}}} \cdot \int_\theta \underbrace{p(D|\theta, m_k)}_{\text{likelihood}} \underbrace{p(\theta|m_k)}_{\text{prior}}\, \mathrm{d}\theta \end{aligned}$$

- You, the engineer, has to choose the factors $p(D|\theta, m_k)$, $p(\theta|m_k)$ and $p(m_k)$. After that, for a given data set D, the model posterior $p(m_k|D)$ can be computed.
- If you need to work with one model,select the model with largest posterior $p(m_k|D)$
- Alternatively, do prediction by **Bayesian model averaging** to utilitize the predictive power from all models:

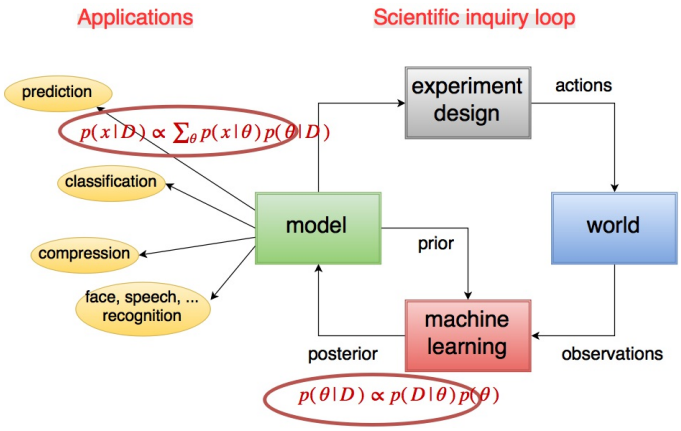
$$\begin{aligned} p(x|D) &= \sum_k \int p(x, \theta, m_k|D)\, \mathrm{d}\theta \\ &= \sum_k \underbrace{p(m_k|D)}_{\substack{\text{model} \\ \text{posterior}}} \cdot \int \underbrace{p(\theta|D)}_{\substack{\text{parameter} \\ \text{posterior}}} \underbrace{p(x|\theta, m_k)}_{\text{likelihood}}\, \mathrm{d}\theta \end{aligned}$$

$\Rightarrow$ Model selection follows the same framework as parameter estimation; it just works at one higher hierarchical level

- More on this in part 2 (Tjalkens).

Machine Learning and the Scientific Method Revisited

- Bayesian probability theory provides a unified framework for information processing (and even the Scientific Method).



Example Problem: Predicting a Coin Toss

- We observe a the following sequence of heads (h) and tails (t) when tossing the same coin repeatedly
$$D = \{hthhtth\}\,.$$
- What is the probability that heads (h) comes up next?

Coin toss example (1): Model Specification

We'll work on the general problem of observing a sequence of N coin tosses $D = \{x_1, \ldots, x_N\}$ with n heads.

Likelihood

- Assume a Bernoulli distributed variable $p(x_n = h|\mu) = \mu$, which leads to a **binomial** distribution for the likelihood:
$$p(D|\mu) = \prod_{n=1}^{N} p(x_n|\mu) = \mu^n (1-\mu)^{N-n}$$

Prior

- Assume the prior belief is governed by a **beta distribution**
$$p(\mu) = \mathcal{B}(\mu|\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\mu^{\alpha-1}(1-\mu)^{\beta-1}$$
  - The Beta distribution is a **conjugate prior** for the Binomial distribution, which means that
$$\text{beta} \propto \text{binomial} \times \text{beta}$$
  - α and β are called **hyperparameters**, since they parameterize the distribution for another parameter (μ). E.g., $\alpha = \beta = 1$ (uniform), $\alpha = \beta = 0.5$ (Jeffreys prior), $\alpha = \beta = 0$ (improper Laplace prior)
  - If α, β are integers, then $\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)(\Gamma(\beta)} = \frac{(\alpha+\beta)!}{\alpha!\,\beta!}$

Coin toss example (2): Parameter estimation

- Infer posterior PDF over μ through Bayes rule
$$\begin{aligned} p(\mu|D) &\propto p(D|\mu)\, p(\mu|\alpha,\beta) \\ &= \mu^n(1-\mu)^{N-n} \times \mu^{\alpha-1}(1-\mu)^{\beta-1} \\ &= \mu^{n+\alpha-1}(1-\mu)^{N-n+\beta-1} \end{aligned}$$

hence the posterior is also beta distributed as
$$p(\mu|D) = \mathcal{B}(\mu|\, n+\alpha, N-n+\beta)$$

- Essentially, **here ends the machine learning activity**

Coin Toss Example (3): Prediction

- Marginalize over the parameter posterior to get the predictive PDF for a new coin toss $x_\bullet$, given the data D,
$$\begin{aligned} p(x_\bullet = h|D) &= \int_0^1 p(x_\bullet = h|\mu)\, p(\mu|D)\, \mathrm{d}\mu \\ &= \int_0^1 \mu \times \mathcal{B}(\mu|\, n+\alpha, N-n+\beta)\, \mathrm{d}\mu \\ &= \frac{n+\alpha}{N+\alpha+\beta} \qquad \text{(a.k.a. Laplace rule)} \end{aligned}$$
- Now we're ready to solve our example problem: for $D = \{hthhtth\}$ and uniform prior ($\alpha = \beta = 1$), we get
$$p(x_\bullet = h|D) = \frac{n+1}{N+2} = \frac{4+1}{7+2} = \frac{5}{9}$$

Coin Toss Example: What did we learn?

- What did we learn from the data? Before seeing any data, we think that

$$p(x_\bullet = h) = p(x_\bullet = h|D)|_{n=N=0} = \frac{\alpha}{\alpha+\beta}\,.$$

- After the N coin tosses, we think that

$$p(x_\bullet = h|D) = \frac{n+\alpha}{N+\alpha+\beta}\,.$$

- Note the following decomposition

$$\begin{aligned} p(x_\bullet = h|\,D) &= \frac{n+\alpha}{N+\alpha+\beta} \\ &= \frac{n}{N+\alpha+\beta} + \frac{\alpha}{N+\alpha+\beta} \\ &= \frac{N}{N+\alpha+\beta}\cdot\frac{n}{N} + \frac{\alpha+\beta}{N+\alpha+\beta}\cdot\frac{\alpha}{\alpha+\beta} \\ &= \underbrace{\frac{\alpha}{\alpha+\beta}}_{prior} + \underbrace{\frac{N}{N+\alpha+\beta}}_{gain}\cdot\Big(\underbrace{\frac{n}{N}}_{MLE} - \underbrace{\frac{\alpha}{\alpha+\beta}}_{prior}\Big) \end{aligned}$$

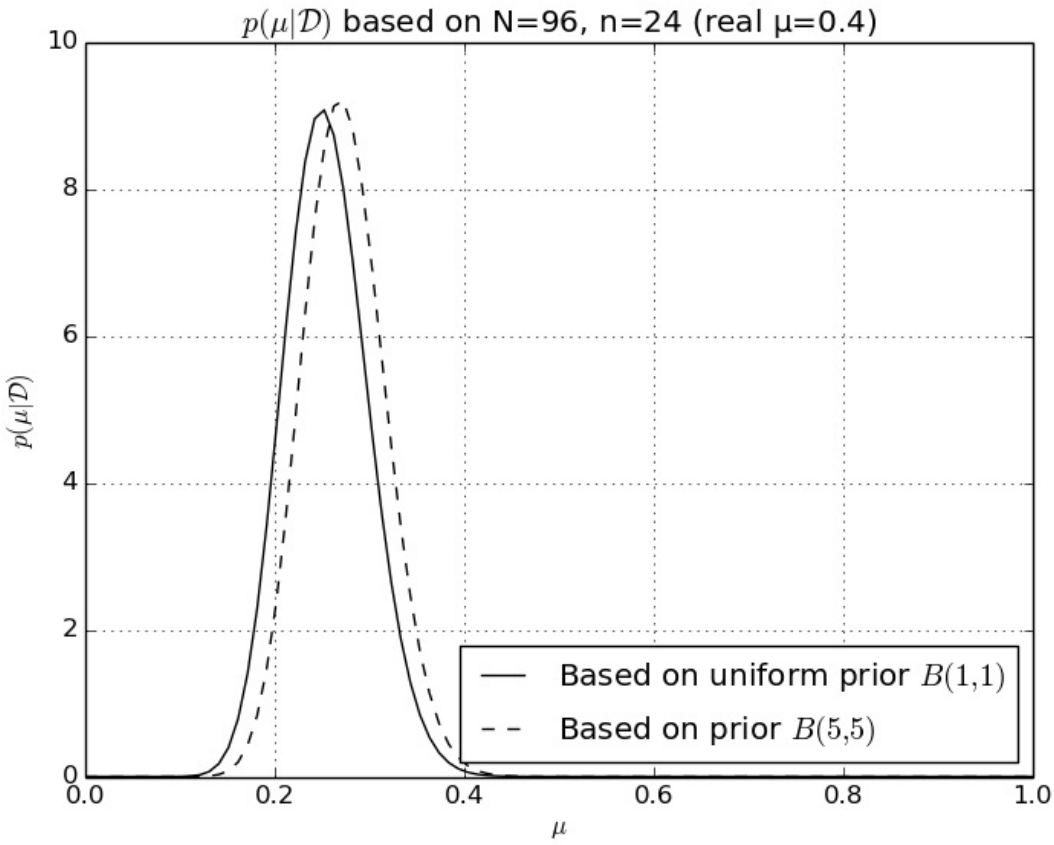
- Note that, since $0 \leq \text{gain} < 1$, the Bayesian estimate lies between prior and maximum likelihood estimate.
- For large N, the gain goes to 1 and $p(x_\bullet = h|D)$ goes to the maximum likelihood estimate (the relative frequency) n/N.

CODE EXAMPLE

Bayesian evolution of $p(\mu|D)$ for the coin toss

Let's see how $p(\mu|D)$ evolves as we increase the number of coin tosses N. We'll use two different priors to demonstrate the effect of the prior on the posterior (set $N = 0$ to inspect the prior).

```
using Reactive, Interact, PyPlot, Distributions
f = figure()
range = linspace(0,1,100)
μ = 0.4
samples = rand(192) .<= μ # Flip 192 coins
@manipulate for N=0:1:192; withfig(f) do
        n = sum(samples[1:N]) # Count number of heads in first N flips
        posterior1 = Beta(1+n, 1+(N-n))
        posterior2 = Beta(5+n, 5+(N-n))
        plot(range, pdf(posterior1,range), "k-")
        plot(range, pdf(posterior2,range), "k--")
        xlabel(L"\mu"); ylabel(L"p(\mu|\mathcal{D})"); grid()
        title(L"p(\mu|\mathcal{D})"*" based on N=$(N), n=$(n) (real μ=$(μ))")
        legend(["Based on uniform prior "*L"B(1,1)","Based on prior "*L"B(5,5)"], loc=4)
    end
end
```



⇒ **With more data, the relevance of the prior diminishes**.

END OF CODE EXAMPLE

From Posterior to Point-Estimate

Sometimes we want just one 'best' parameter (vector), rather than a posterior distribution over parameters. Why?

- Recall Bayesian prediction

$$p(x|D) = \int p(x|\theta)p(\theta|D)\,\mathrm{d}\theta$$

- If we approximate posterior $p(\theta|D)$ by a delta function for one 'best' value $\hat{\theta}$, then the predictive distribution collapses to

$$p(x|D) = \int p(x|\theta)\,\delta(\theta-\hat{\theta})\,\mathrm{d}\theta = p(x|\hat{\theta})$$

- This is the model $p(x|\theta)$ evaluated at $\theta = \hat{\theta}$.
- Note that $p(x|\hat{\theta})$ is much easier to evaluate than the integral for full Bayesian prediction.

Some Well-known Point-Estimates

1. **Bayes estimate**

$$\hat{\theta}_{bayes} = \int \theta\, p(\theta|D)\,\mathrm{d}\theta$$

   - (homework). Proof that the Bayes estimate minimizes the expected mean-square error, i.e., proof that

$$\hat{\theta}_{bayes} = \underset{\hat{\theta}}{\operatorname{argmin}} \int_{\theta} (\hat{\theta}-\theta)^2 p(\theta|D)\,\mathrm{d}\theta$$

2. **Maximum A Posteriori** (MAP) estimate

$$\hat{\theta}_{\mathrm{map}} = \underset{\theta}{\operatorname{argmax}}\, p(\theta|D) = \underset{\theta}{\operatorname{argmax}} p(D|\theta)\; p(\theta)$$

3. **Maximum Likelihood** (ML) estimate

$$\hat{\theta}_{ml} = \underset{\theta}{\operatorname{arg\,max}} p(D|\theta)$$

   - Note that Maximum Likelihood is MAP with uniform prior

Bayesian vs Maximum Likelihood Learning

Consider the task: predict a datum x from an observed data set D.

	Bayesian	Maximum Likelihood
1. **Model Specification**	Choose a model m with data generating distribution $p(x\|\theta,m)$ and parameter prior $p(\theta\|m)$	Choose a model m with same data generating distribution $p(x\|\theta,m)$. No need for priors.
2. **Learning**	use Bayes rule to find the parameter posterior, $p(\theta\|D) = \propto p(D\|\theta)p(\theta)$	By Maximum Likelihood (ML) optimization, $\hat{\theta} = \arg\max_{\theta} p(D\|\theta)$
3. **Prediction**	$p(x\|D) = \int p(x\|\theta)p(\theta\|D)\,\mathrm{d}\theta$	$p(x\|D) = p(x\|\hat{\theta})$

Report Card on Maximum Likelihood Estimation

- Maximum Likelihood (ML) is MAP with uniform prior, or MAP is 'penalized' ML

$$\hat{\theta}_{map} = \arg\max_{\theta} \{ \overbrace{\log p\left(D|\theta\right)}^{\text{log-likelihood}} + \overbrace{\log p\left(\theta\right)}^{\text{penalty}} \}$$

- (good!). Works rather well if we have a lot of data because the influence of the prior diminishes with more data.
- (bad). Cannot be used for model comparison. E.g. best model does generally not correspond to largest likelihood (see part-2, Tjalkens).
- (good). Computationally often do-able. Useful fact (since $\log$ is monotonously increasing):

$$\arg\max_{\theta} \log p(D|\theta) = \arg\max_{\theta} p(D|\theta)$$

$\Rightarrow$ **ML estimation is an approximation to Bayesian learning**, but in the face of lots of available data for good reason a very popular learning method.

Working with Gaussians

Preliminaries

- Goal
  - Review of processing of Gaussian distributions in linear systems
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 85-93
    - MacKay - 2006 - The Humble Gaussian Distribution

Sums and Transformations of Gaussian Variables

- The Gaussian distribution

$$\mathcal{N}(x|\mu,\Sigma) = |2\pi\Sigma|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\}$$

  for variable x is completely specified by its mean μ and variance Σ.
- $\Lambda = \Sigma^{-1}$ is called the **precision matrix**.
- A **linear transformation** $z = Ax + b$ of a Gaussian variable $\mathcal{N}(x|\mu,\Sigma)$ is Gaussian distributed as

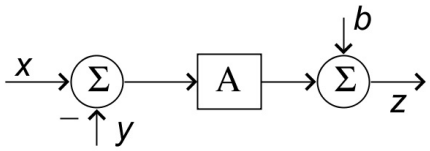
$$p(z) = \mathcal{N}\left(z \mid A\mu + b, A\Sigma A^T\right) \tag{SRG-4a}$$

- The **sum of two independent Gaussian variables** is also Gaussian distributed. Specifically, if $x \sim \mathcal{N}(x|\mu_x,\Sigma_x)$ and $y \sim \mathcal{N}(y|\mu_y,\Sigma_y)$, then the PDF for $z = x + y$ is given by

$$\begin{aligned} p(z) &= \mathcal{N}(x \mid \mu_x, \Sigma_x) * \mathcal{N}(y \mid \mu_y, \Sigma_y) \\ &= \mathcal{N}\left(z \mid \mu_x + \mu_y, \Sigma_x + \Sigma_y\right) \end{aligned} \tag{SRG-8}$$

  - [Exercise]: Show that Eq.SRG-8 is really a special case of Eq.SRG-4a.
- The sum of two Gaussian *distributions* is NOT a Gaussian distribution. Why not?

Example: Gaussian Signals in a Linear System



- [Q.]: Given independent variables $x \sim \mathcal{N}(\mu_x,\sigma_y)$ and $y \sim \mathcal{N}(\mu_y,\sigma_y)$, what is the PDF for $z = A \cdot (x - y) + b$?
- [A.]: z is also Gaussian with

$$p_z(z) = \mathcal{N}(z|A(\mu_x - \mu_y) + b,\, A(\sigma_x + \sigma_y)A^T)$$

- Think about the role of the Gaussian distribution for stochastic linear systems in relation to what sinusoidals mean for deterministic linear system analysis.

Example: Bayesian Estimation of a Constant

[Question]

- Estimate a constant θ from one 'noisy' measurement x about that constant. Assume the following model specification:

$$\begin{aligned} x &= \theta + \epsilon \\ \theta &\sim \mathcal{N}(\theta|\mu_\theta, \sigma_\theta^2) \\ \epsilon &\sim \mathcal{N}(\epsilon|0, \sigma_\epsilon^2) \end{aligned}$$

[Answer]

1. **Model specification** Note that you can rewrite these specifications in probabilistic notation as follows:

$$\begin{aligned} p(x|\theta) &= \mathcal{N}(x|\theta, \sigma_\epsilon^2) && \text{(likelihood)} \\ p(\theta) &= \mathcal{N}(\theta|\mu_\theta, \sigma_\theta^2) && \text{(prior)} \end{aligned}$$

2. **Inference** for the posterior PDF $p(\theta|x)$

$$\begin{aligned} p(\theta|x) &= \frac{p(x|\theta)p(\theta)}{p(x)} = \frac{p(x|\theta)p(\theta)}{\int p(x|\theta)p(\theta)\,\mathrm{d}\theta} \\ &= \frac{1}{C}\mathcal{N}(x|\theta, \sigma_\epsilon^2)\,\mathcal{N}(\theta|\mu_\theta, \sigma_\theta^2) \\ &= \frac{1}{C_1}\exp\left\{-\frac{(x-\theta)^2}{2\sigma_\epsilon^2} - \frac{(\theta-\mu_\theta)^2}{2\sigma_\theta^2}\right\} \\ &= \frac{1}{C_1}\exp\left\{\theta^2\left(-\frac{1}{2\sigma_\epsilon^2} - \frac{1}{2\sigma_\theta^2}\right) + \theta\left(\frac{x}{\sigma_\epsilon^2} + \frac{\mu_\theta}{\sigma_\theta^2}\right) + C_2\right\} \\ &= \frac{1}{C_1}\exp\left\{-\frac{\sigma_\theta^2 + \sigma_\epsilon^2}{2\sigma_\theta^2\sigma_\epsilon^2}\left(\theta - \frac{x\sigma_\theta^2 + \mu_s\sigma_\epsilon^2}{\sigma_\theta^2 + \sigma_\epsilon^2}\right)^2 + C_3\right\} \end{aligned}$$

   which we recognize as a Gaussian distribution.

   - This computational 'trick' for multiplying two Gaussians is called **completing the square**. Compare the procedure to

$$ax^2 + bx + c_1 = a\left(x + \frac{b}{2a}\right)^2 + c_2$$

- Hence, it follows that the posterior for θ is

$$p(\theta|x) = \mathcal{N}(\theta|\,\mu_{\theta|x}, \sigma_{\theta|x}^2)$$

  where

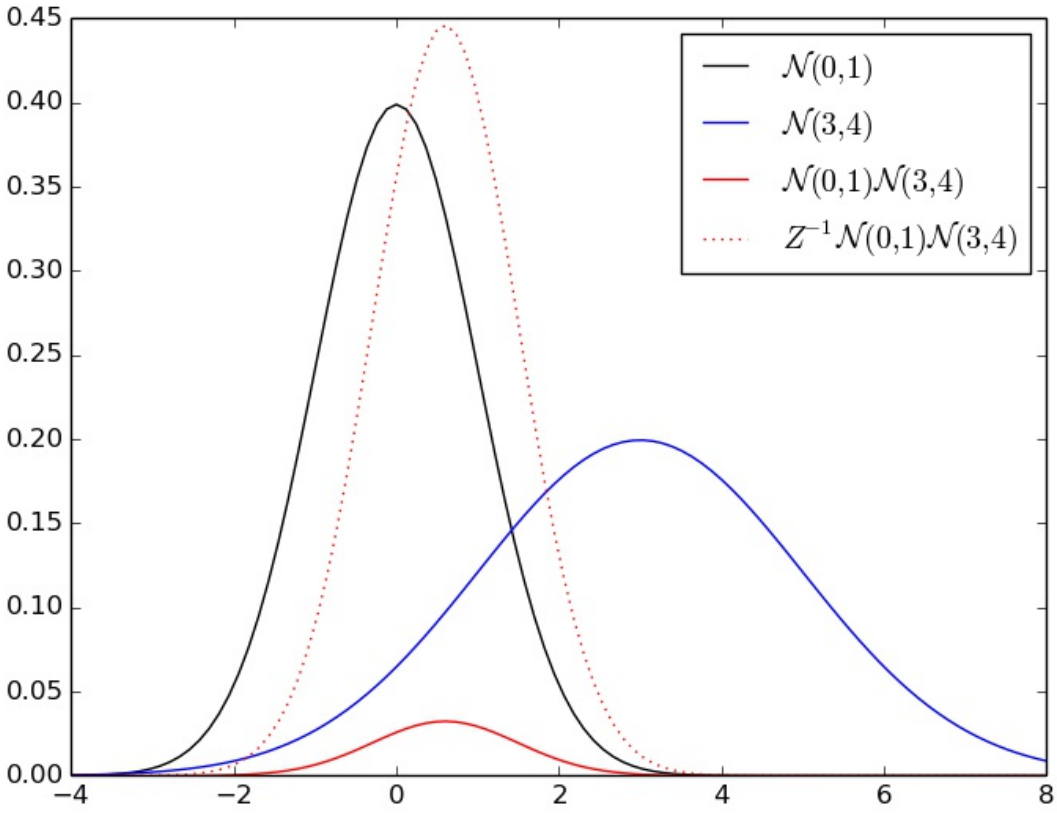
$$\begin{aligned} \sigma_{\theta|x}^2 &= \frac{\sigma_\epsilon^2\sigma_\theta^2}{\sigma_\epsilon^2 + \sigma_\theta^2} = \left(\frac{1}{\sigma_\theta^2} + \frac{1}{\sigma_\epsilon^2}\right)^{-1} \\ \mu_{\theta|x} &= \sigma_{\theta|x}^2\left(\frac{1}{\sigma_\epsilon^2}x + \frac{1}{\sigma_\theta^2}\mu_\theta\right) \end{aligned}$$

- So, multiplication of two Gaussians yields another (unnormalized) Gaussian.

CODE EXAMPLE

Let's plot the exact product of two Gaussian PDFs as well as the normalized product according to the above derivation.

```
using PyPlot, Distributions
d1 = Normal(0, 1) # μ=0, σ^2=1
d2 = Normal(3, 2) # μ=3, σ^2=4

# Calculate the parameters of the product d1*d2
s2_prod = (d1.σ^-2 + d2.σ^-2)^-1
m_prod = s2_prod * ((d1.σ^-2)*d1.μ + (d2.σ^-2)*d2.μ)
d_prod = Normal(m_prod, sqrt(s2_prod)) # Note that we neglect the normalization constant.

# Plot stuff
x = linspace(-4, 8, 100)
plot(x, pdf(d1,x), "k")
plot(x, pdf(d2,x), "b")
plot(x, pdf(d1,x) .* pdf(d2,x), "r-") # Plot the exact product
plot(x, pdf(d_prod,x), "r:")          # Plot the normalized Gaussian product
legend([L"\mathcal{N}(0,1)",
        L"\mathcal{N}(3,4)",
        L"\mathcal{N}(0,1) \mathcal{N}(3,4)",
        L"Z^{-1} \mathcal{N}(0,1) \mathcal{N}(3,4)"]);
```



The solid and dotted red curves are identical up to a scaling factor Z.

END OF CODE EXAMPLE

Multivariate Gaussian Multiplication

- In general, the multiplication of two multi-variate Gaussians yields an (unnormalized) Gaussian, see [SRG-6]:

$$\mathcal{N}(x|\mu_a, \Sigma_a) \cdot \mathcal{N}(x|\mu_b, \Sigma_b) = \alpha \cdot \mathcal{N}(x|\mu_c, \Sigma_c)$$

  where

$$\Sigma_c = \left(\Sigma_a^{-1} + \Sigma_b^{-1}\right)^{-1}$$
$$\mu_c = \Sigma_c \left(\Sigma_a^{-1}\mu_a + \Sigma_b^{-1}\mu_b\right)$$

  and normalization constant $\alpha = \mathcal{N}(\mu_a|\, \mu_b, \Sigma_a + \Sigma_b)$.

- If we define the **precision** as $\Lambda \equiv \Sigma^{-1}$, then we see that **precisions add** and **precision-weighted means add** too.

- As we just saw, great application to Bayesian inference!

$$\underbrace{\text{Gaussian}}_{\text{posterior}} \propto \underbrace{\text{Gaussian}}_{\text{likelihood}} \times \underbrace{\text{Gaussian}}_{\text{prior}}$$

Conditioning and Marginalization of a Gaussian

Let $z = \begin{bmatrix} x \\ y \end{bmatrix}$ be jointly normal distributed as

$$p(z|\mu, \Sigma) = \mathcal{N}\left(\begin{bmatrix} x \\ y \end{bmatrix} \middle| \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} \Sigma_x & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_y \end{bmatrix} \right)$$

- Note that the symmetry $\Sigma = \Sigma^T$ implies that Σ_x and Σ_y are symmetric and $\Sigma_{xy} = \Sigma_{yx}^T$.
- Now let's factorize $p(x, y)$ into $p(y|x)\, p(x)$ through conditioning and marginalization (for applications to Bayesian inference in jointly Gaussian systems).
- **Marginalization**

$$p(x) = \int p(x, y)\, \mathrm{d}y = \mathcal{N}\left(x|\mu_x, \Sigma_x\right)$$

- **Conditioning**

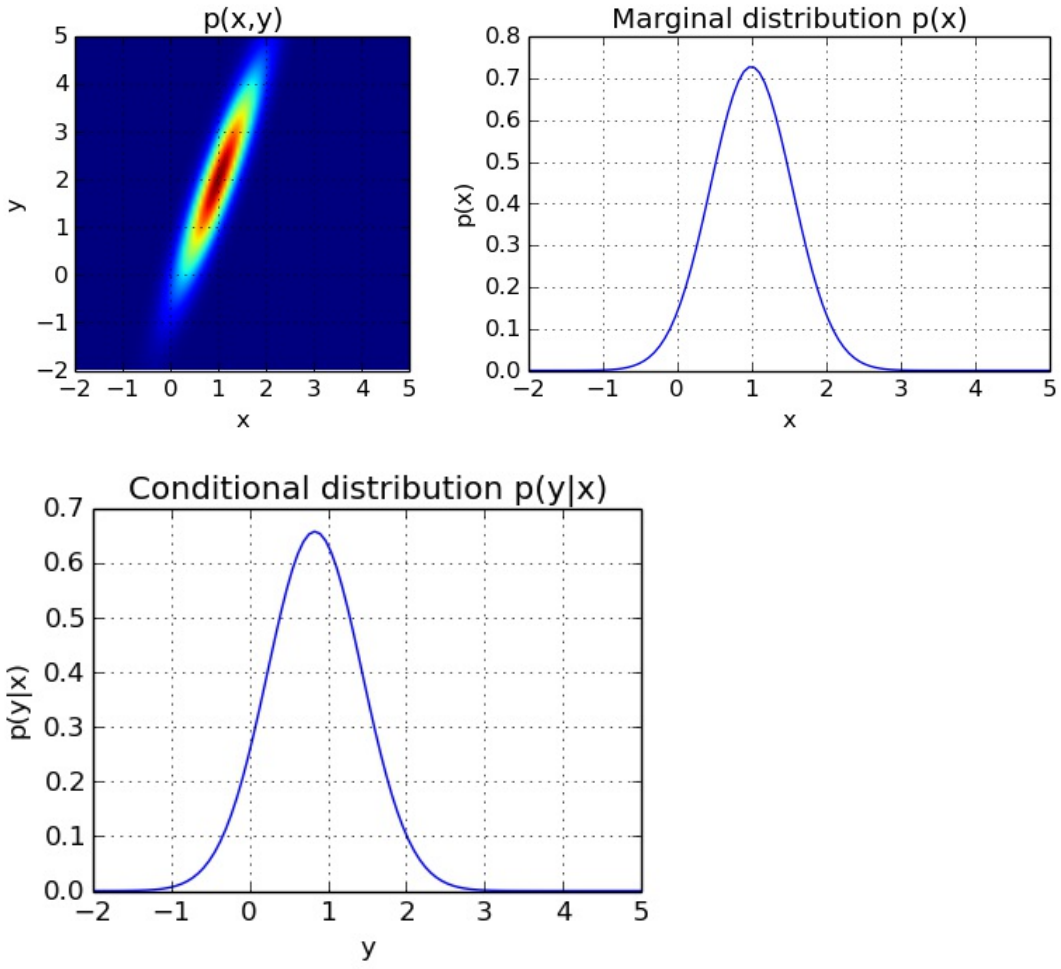
$$\begin{aligned} p(y|x) &= p(x, y)/p(x) \\ &= \mathcal{N}\left(y|\mu_y + \Sigma_{yx}\Sigma_x^{-1}(x - \mu_x),\, \Sigma_y - \Sigma_{yx}\Sigma_x^{-1}\Sigma_{xy}\right) \end{aligned}$$

- **Exercise**: Try to understand these formulas

CODE EXAMPLE

Interactive plot of the joint, marginal, and conditional distributions.

```
using Reactive, Interact, PyPlot, Distributions
# z = [x; y]
μ = [1.; 2.]
Σ = [0.3 0.7;
     0.7 2.0]
joint = MvNormal(μ,Σ)
marginal_x = Normal(μ[1], sqrt(Σ[1,1]))

# Plot p(x,y)
subplot(221)
joint_pdf = Matrix(100,100)
x_range = linspace(-2,5,100); y_range = linspace(-2,5,100)
for i=1:length(x_range)
    for j=1:length(y_range)
        joint_pdf[i,j] = pdf(joint, [x_range[i];y_range[j]])
    end
end

imshow(joint_pdf', origin="lower", extent=[x_range[1], x_range[end], y_range[1], y_range[end]])
grid(); xlabel("x"); ylabel("y"); title("p(x,y)"); tight_layout()

# Plot p(x)
subplot(222)
plot(linspace(-2,5,100), pdf(marginal_x, linspace(-2,5,100)))
grid(); xlabel("x"); ylabel("p(x)"); title("Marginal distribution p(x)"); tight_layout()

f = figure()
@manipulate for x=-2:0.1:3; withfig(f) do
        conditional_y_m = μ[2]+Σ[2,1]*inv(Σ[1,1])*(x-μ[1])
        conditional_y_s2 = Σ[2,2] - Σ[2,1]*inv(Σ[1,1])*Σ[1,2]
        conditional_y = Normal(conditional_y_m, sqrt(conditional_y_s2))

        # Plot p(y|x)
        subplot(223)
        plot(linspace(-2,5,100), pdf(conditional_y, linspace(-2,5,100)))
        grid(); xlabel("y"); ylabel("p(y|x)"); title("Conditional distribution p(y|x)"); tight_lay
out()
    end
end
```



As is clear from the plots, the conditional distribution is a renormalized slice from the joint distribution.

END OF CODE EXAMPLE

Example: Conditioning of Gaussian

Consider (again) the system

$$
\begin{aligned}
x &= \theta + \epsilon \\
\theta &\sim \mathcal{N}(\theta|\mu_\theta, \sigma_\theta^2) \\
\epsilon &\sim \mathcal{N}(\epsilon|0, \sigma_\epsilon^2)
\end{aligned}
$$

- This system is equivalent to (Exercise: derive this!)

$$
p(x, \theta|\, \mu, \sigma) = \mathcal{N}\left(\begin{bmatrix} x \\ \theta \end{bmatrix} \middle| \begin{bmatrix} \mu_\theta \\ \mu_\theta \end{bmatrix}, \begin{bmatrix} \sigma_\theta^2 + \sigma_\epsilon^2 & \sigma_\theta^2 \\ \sigma_\theta^2 & \sigma_\theta^2 \end{bmatrix} \right)
$$

- Direct substitution of the rule for Gaussian conditioning leads to

$$
p(\theta|x) = \mathcal{N}\left(\theta \,|\, \mu_{\theta|x}, \sigma_{\theta|x}^2\right)
$$

with

$$
\begin{aligned}
K &= \frac{\sigma_\theta^2}{\sigma_\theta^2 + \sigma_\epsilon^2} \qquad (K \text{ is called: Kalman gain}) \\
\mu_{\theta|x} &= \mu_\theta + K \cdot (x - \mu_\theta) \\
\sigma_{\theta|x}^2 &= (1 - K)\sigma_\theta^2
\end{aligned}
$$

- Homework exercises: (1) Actually derive this; (2) show that the result is equivalent to the previous slide on 'estimation of a constant'; and (3) Try to interpret the resulting formula's.

⟶ Moral: For jointly Gaussian systems, we do inference simply in one step by using the formulas for conditioning and marginalization.

Application: Recursive Bayesian Estimation

Now consider the signal $x_t = \theta + \epsilon_t$, where $D_t = \{x_1, \ldots, x_t\}$ is observed *sequentially* (over time).

[Question]

- Derive a recursive algorithm for $p(\theta|D_t)$, i.e., an update rule for (posterior) $p(\theta|D_t)$ based on (prior) $p(\theta|D_{t-1})$ and (new observation) x_t.

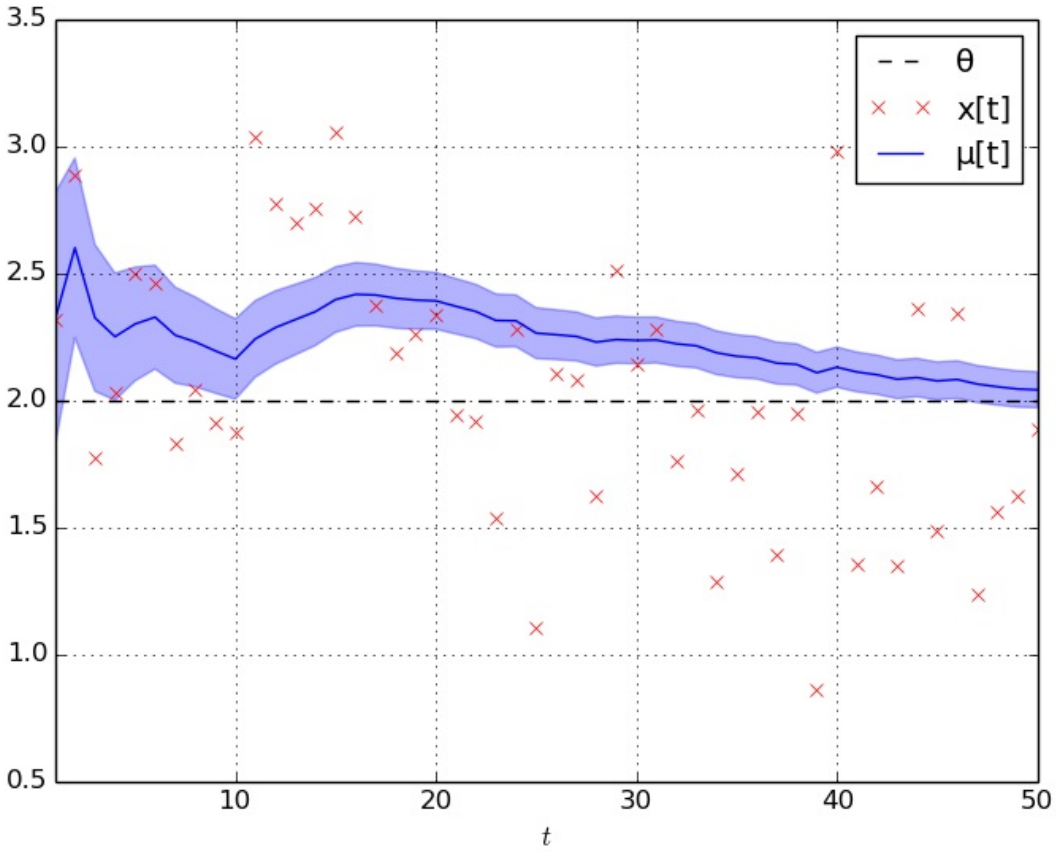
[Answer]

- **Model specification**. Let's define the estimate after t observations as
$$p(\theta|D_t) = \mathcal{N}(\theta \,|\, \mu_t, \sigma_t^2).$$
We will use the same likelihood as before:
$$p(x_t|\theta) = \mathcal{N}(x_t|\theta, \sigma_\epsilon^2)$$
- **Inference**. We will solve this by using the estimate after $t-1$ as the **prior distribution** in conjunction with the **likelihood** for observation $x(t)$,
$$\begin{aligned} p(\theta|D_t) &\propto p(x_t|\theta)\, p(\theta|D_{t-1}) \\ &= \mathcal{N}(x_t|\theta, \sigma_\epsilon^2)\, \mathcal{N}(\theta \,|\, \mu_{t-1}, \sigma_{t-1}^2) \\ &= \mathcal{N}(\theta|x_t, \sigma_\epsilon^2)\, \mathcal{N}(\theta \,|\, \mu_{t-1}, \sigma_{t-1}^2) \\ &= \mathcal{N}(\theta|\mu_t, \sigma_t^2) \end{aligned}$$
with
$$\begin{aligned} K_t &= \frac{\sigma_{t-1}^2}{\sigma_{t-1}^2 + \sigma_\epsilon^2} \qquad (\text{Kalman gain}) \\ \mu_t &= \mu_{t-1} + K_t \cdot (x_t - \mu_{t-1}) \\ \sigma_t^2 &= (1 - K_t)\, \sigma_{t-1}^2 \end{aligned}$$
(as before, we used the formulas for conditioning in a multivariate Gaussian system).
- This linear *sequential* estimator of mean and variance in Gaussian observations is called a **Kalman Filter**.
- The new observation x_t 'updates' the old estimate μ_{t-1} by a quantity that is proportional to the *innovation* (or *residual*) $(x_t - \mu_{t-1})$.
- Note that the uncertainty about θ decreases over time (since $0
- Recursive Bayesian estimation is the basis for **adaptive signal processing** algorithms such as Least Mean Squares (LMS) and Recursive Least Squares (RLS).

CODE EXAMPLE

Let's implement the Kalman filter described above. We'll use it to recursively estimate the value of θ based on noisy observations. Use the 'Step' button to see the recursive updates to the posterior $p(\theta|D)$.

```
using PyPlot, Reactive, Interact

interactive_plot = false # Set to true to generate an interactive plot with 'step' button
N = 50                   # Number of observations
θ = 2.0                  # True value of the variable we want to estimate
σ_ϵ2 = 0.25              # Observation noise variance
x = sqrt(σ_ϵ2) * randn(N) + θ # Generate N noisy observations of θ

f = figure()
global t = 0
global μ = fill!(Vector{Float64}(N), NaN)    # Means of p(θ|D) over time
global σ_μ2 = fill!(Vector{Float64}(N), NaN) # Variances of p(θ|D) over time

function performKalmanStep()
    # Perform a Kalman filter step, update t, μ, σ_μ2
    global t += 1
    if t>1 # Use posterior from prev. step as prior
        K = σ_μ2[t-1] / (σ_ϵ2 + σ_μ2[t-1]) # Kalman gain
        μ[t] = μ[t-1] + K*(x[t] - μ[t-1])  # Update mean using (1)
        σ_μ2[t] = σ_μ2[t-1] * (1.0-K)      # Update variance using (2)
    elseif t==1 # Use prior
        # Prior p(θ) = N(0,1000)
        K = 1000.0 / (σ_ϵ2 + 1000.0) # Kalman gain
        μ[t] = 0 + K*(x[t] - 0)      # Update mean using (1)
        σ_μ2[t] = 1000 * (1.0-K)     # Update variance using (2)
    end
end

function plotStatus()
    # Plot the 'true' value of θ, noisy observations x, and the recursively updated posterior p(θ|
D)
    plot([1:N], θ*ones(N), "k--")
    plot([1:N], x, "rx")
    plot([1:N], μ, "b-")
    fill_between([1:N], μ-sqrt(σ_μ2), μ+
    sqrt(σ_μ2), color="b", alpha=0.3)
    legend(["θ", "x[t]", "μ[t]"])
    xlim([1,N]); xlabel(L"t"); grid()
end

if interactive_plot
    @manipulate for
        perform_step = button("Step");
        withfig(f) do
            if t<=N
                performKalmanStep()
                plotStatus()
            end
        end
    end
else
    while t<N
        performKalmanStep()
    end
    plotStatus()
end
```



The shaded area represents 2 standard deviations of posterior $p(\theta|D)$. The variance of the posterior is guaranteed to decrease monotonically for the standard Kalman filter.

END OF CODE EXAMPLE

Product of Normally Distributed Variables

- (We've seen that) the sum of two Gausssian distributed variables is also Gaussian distributed.
- Has the *product* of two Gaussian distributed variables also a Gaussian distribution?
- **No**! In general this is a difficult computation. As an example, let's compute $p(z)$ for $Z = XY$ for the special case that $X \sim \mathcal{N}(0,1)$ and $Y \sim \mathcal{N}(0,1)$.

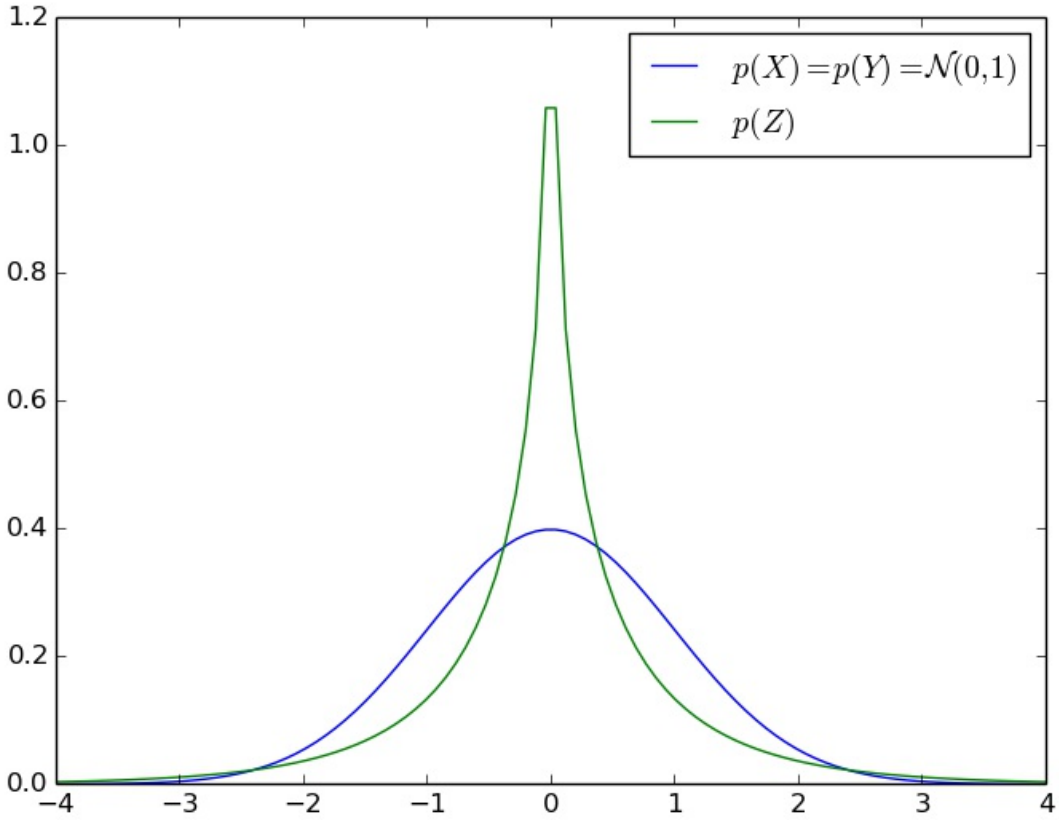
$$\begin{aligned} p(z) &= \int_{X,Y} p(z|x,y)\, p(x,y)\, \mathrm{d}x\mathrm{d}y \\ &= \frac{1}{2\pi} \int \delta(z-xy)\, e^{-(x^2+y^2)/2}\, \mathrm{d}x\mathrm{d}y \\ &= \frac{1}{\pi} \int_0^\infty \frac{1}{x} e^{-(x^2+z^2/x^2)/2}\, \mathrm{d}x \\ &= \frac{1}{\pi} \mathrm{K}_0\left(|z|\right). \end{aligned}$$

where $\mathrm{K}_n(z)$ is a modified Bessel function of the second kind.

CODE EXAMPLE

We plot $p(Z)$ to give an idea of what this distribution looks like.

```
using PyPlot, Distributions
X = Normal(0,1)
pdf_product_std_normals(z::Vector) = (besselk(0, abs(z))./π)
range = collect(linspace(-4,4,100))
plot(range, pdf(X, range))
plot(range, pdf_product_std_normals(range))
legend([L"p(X)=p(Y)=\mathcal{N}(0,1)", L"p(Z)"]);
```



END OF CODE EXAMPLE

Review Gaussians

The success of Gaussian distributions in probabilistic modeling is large due to the following properties:

- The convolution of two Gaussian functions is another Gaussian function (use in sum of 2 variables)
- The product of two Gaussian functions is another Gaussian function (use in Bayes rule).
- A linear transformation of a Gaussian distributed variable is also Gaussian distributed
- Conditioning and marginalization of multivariate Gaussian distributions produce Gaussians again (use in working with observations and when doing Bayesian predictions)
- The Gaussian PDF has higher entropy than any other with the same variance. (Not discussed in this course).
- Any smooth function with single rounded maximum, if raised to higher and higher powers, goes into a Gaussian function. (Not discussed).

Some Useful Matrix Calculus

Aside from working with Gaussians, it will be helpful for the next lessons to be familiar with some matrix calculus. We shortly recapitulate used formulas here.

- We define the **gradient** of a scalar function $f(A)$ w.r.t. an $n \times k$ matrix A as

$$\nabla_A f \triangleq \begin{bmatrix} \frac{\partial f}{\partial a_{11}} & \frac{\partial f}{\partial a_{12}} & \cdots & \frac{\partial f}{\partial a_{1k}} \\ \frac{\partial f}{\partial a_{21}} & \frac{\partial f}{\partial a_{22}} & \cdots & \frac{\partial f}{\partial a_{2k}} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial f}{\partial a_{n1}} & \frac{\partial f}{\partial a_{n2}} & \cdots & \frac{\partial f}{\partial a_{nk}} \end{bmatrix}$$

- The following formulas are useful (see Bishop App.-C)

$$\begin{aligned} |A^{-1}| &= |A|^{-1} && \text{(B-C.4)} \\ \nabla_A \log|A| &= (A^T)^{-1} = (A^{-1})^T && \text{(B-C.28)} \\ \mathrm{Tr}[ABC] &= \mathrm{Tr}[CAB] = \mathrm{Tr}[BCA] && \text{(B-C.9)} \\ \nabla_A \mathrm{Tr}[AB] &= \nabla_A \mathrm{Tr}[BA] = B^T && \text{(B-C.25)} \\ \nabla_A \mathrm{Tr}[ABA^T] &= A(B+B^T) && \text{(B-C.27)} \\ \nabla_x x^T Ax &= (A+A^T)x && \text{(from B-C.27)} \\ \nabla_X a^T Xb &= \nabla_X \mathrm{Tr}[ba^T X] = ab^T \end{aligned}$$

What's Next?

- We discussed how Bayesian probability theory provides an integrated framework for making predictions based on observed data.
- The process involves model specification (your main task!), inference and actual model-based prediction.
- The latter two tasks are only difficult because of computational issues.
- Maximum likelihood was introduced as a computationally simpler approximation to the Bayesian approach.
- In particular under some linear Gaussian assumptions, a few interesting models can be designed.
- The rest of this course (part-1) concerns introduction to these Linear Gaussian models.

Density Estimation

Preliminaries

- Goal
  - Simple maximum likelihood estimates for Gaussian and categorical distributions
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 67-70, 74-76, 93-94

Why Density Estimation?

Density estimation relates to building a model $p(x|\theta)$ from observations $D = \{x_1, \ldots, x_N\}$.

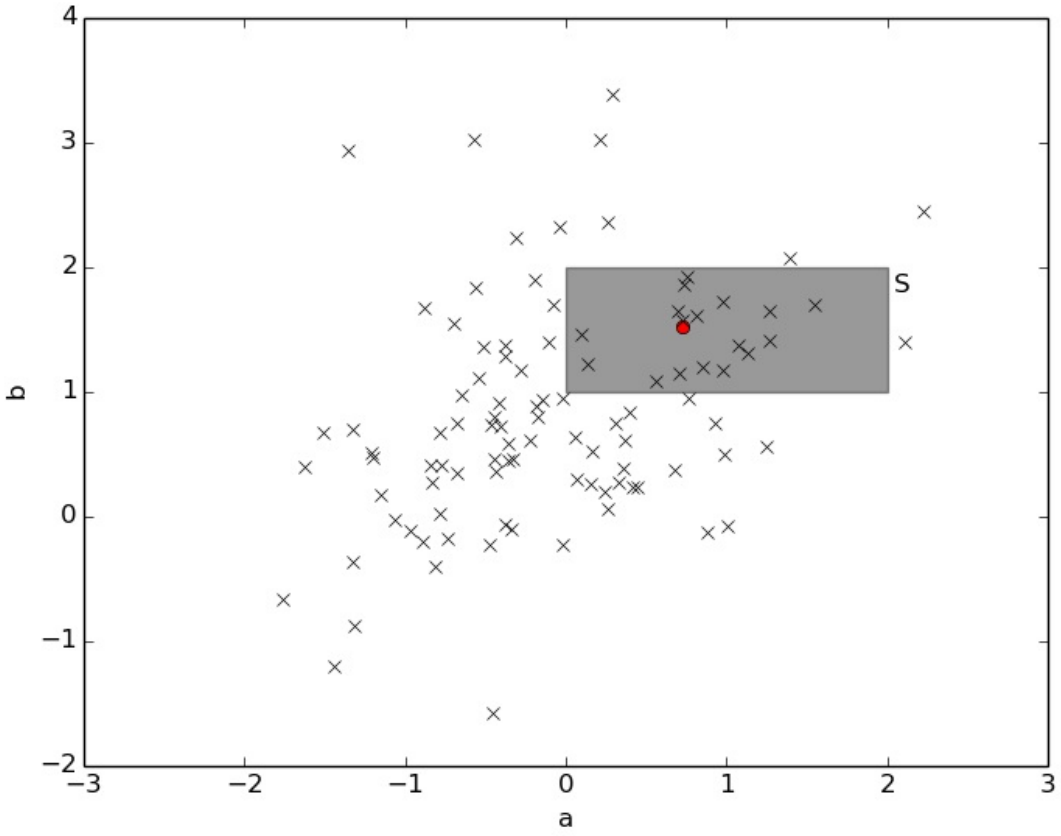
Why is this interesting? Some examples:

- **Outlier detection**. Suppose $D = \{x_n\}$ are benign mammogram images. Build $p(x|\theta)$ from D. Then low value for $p(x'|\theta)$ indicates that x' is a risky mammogram.
- **Compression**. Code a new data item based on **entropy**, which is a functional of $p(x|\theta)$:
$$H[p] = -\sum_x p(x|\theta) \log p(x|\theta)$$
- **Classification**. Let $p(x|\theta_1)$ be a model of attributes x for credit-card holders that paid on time and $p(x|\theta_2)$ for clients that defaulted on payments. Then, assign a potential new client x' to either class based on the relative probability of $p(x'|\theta_1)$ vs. $p(x'|\theta_2)$.

Example Problem

Consider a set of observations $D = \{x_1, \ldots, x_N\}$ in the 2-dimensional plane (see Figure). All observations were generated by the same process. We now draw an extra observation $x_\bullet = (a, b)$ from the same data generating process. What is the probability that $x_\bullet$ lies within the shaded rectangle S?

```
using Distributions, PyPlot
N = 100
generative_dist = MvNormal([0,1.], [0.8 0.5; 0.5 1.0])
function plotObservations(obs::Matrix)
    plot(obs[1,:]', obs[2,:]', "kx", zorder=3)
    fill_between([0., 2.], 1., 2., color="k", alpha=0.4, zorder=2) # Shaded area
    text(2.05, 1.8, "S", fontsize=12)
    xlim([-3,3]); ylim([-2,4]); xlabel("a"); ylabel("b")
end
D = rand(generative_dist, N) # Generate observations from generative_dist
plotObservations(D)
x_dot = rand(generative_dist) # Generate x•
plot(x_dot[1], x_dot[2], "ro")
```




```
1-element Array{Any,1}:
 PyObject <matplotlib.lines.Line2D object at 0x7fbc949b6668>
```

Log-Likelihood for a Multivariate Gaussian (MVG)

- Assume we are given a set of IID data points $D = \{x_1, \ldots, x_N\}$, where $x_n \in \mathbb{R}^D$. We want to build a model for these data.
- **Model specification**. Let's assume a MVG model $x_n = \mu + \epsilon_n$ with $\epsilon_n \sim \mathcal{N}(0, \Sigma)$, or equivalently,

$$\begin{aligned} p(x_n|\mu, \Sigma) &= \mathcal{N}(x_n|\mu, \Sigma) \\ &= |2\pi\Sigma|^{-1/2} \exp\left\{-\frac{1}{2}(x_n - \mu)^T \Sigma^{-1}(x_n - \mu)\right\} \end{aligned}$$

- Since the data are IID, $p(D|\theta)$ factorizes as

$$p(D|\theta) = p(x_1, \ldots, x_N|\theta) \overset{\text{IID}}{=} \prod_n p(x_n|\theta)$$

- This choice of model yields the following log-likelihood (use (B-C.9) and (B-C.4)),

$$\begin{aligned} \log p(D|\theta) &= \log \prod_n p(x_n|\theta) \\ &= \sum_n \log \mathcal{N}(x_n|\mu, \Sigma) \\ &= N \cdot \log|2\pi\Sigma|^{-1/2} - \frac{1}{2}\sum\nolimits_n (x_n - \mu)^T \Sigma^{-1}(x_n - \mu) \end{aligned} \tag{1}$$

Maximum Likelihood estimation of mean of MVG

We want to maximize $\log p(D|\theta)$ wrt the parameters $\theta = \{\mu, \Sigma\}$. Let's take derivatives; first to mean μ, (making use of (B-C.25) and (B-C.27)),

$$\begin{aligned}\nabla_\mu \log p(D|\theta) &= -\frac{1}{2}\sum_n \nabla_\mu \left[(x_n-\mu)^T\Sigma^{-1}(x_n-\mu)\right] \\ &= -\frac{1}{2}\sum_n \nabla_\mu \mathrm{Tr}\left[-2\mu^T\Sigma^{-1}x_n + \mu^T\Sigma^{-1}\mu\right] \\ &= -\frac{1}{2}\sum_n \left(-2\Sigma^{-1}x_n + 2\Sigma^{-1}\mu\right) \\ &= \Sigma^{-1}\sum_n (x_n-\mu)\end{aligned}$$

Set to zero yields the **sample mean**

$$\hat{\mu} = \frac{1}{N}\sum_n x_n$$

Maximum Likelihood estimation of variance of MVG

Now we take the gradient of the log-likelihood **wrt the precision matrix** Σ^{-1} (making use of B-C.28 and B-C.24)

$$\begin{aligned}&\nabla_{\Sigma^{-1}} \log p(D|\theta) \\ &= \nabla_{\Sigma^{-1}}\left[\frac{N}{2}\log|2\pi\Sigma|^{-1} - \frac{1}{2}\sum_{n=1}^N (x_n-\mu)^T\Sigma^{-1}(x_n-\mu)\right] \\ &= \nabla_{\Sigma^{-1}}\left[\frac{N}{2}\log|\Sigma^{-1}| - \frac{1}{2}\sum_{n=1}^N \mathrm{Tr}\left[(x_n-\mu)(x_n-\mu)^T\Sigma^{-1}\right]\right] \\ &= \frac{N}{2}\Sigma - \frac{1}{2}\sum_n (x_n-\mu)(x_n-\mu)^T\end{aligned}$$

Get optimum by setting the gradient to zero,

$$\hat{\Sigma} = \frac{1}{N}\sum_n (x_n-\hat{\mu})(x_n-\hat{\mu})^T$$

which is the **sample variance**.

(OPTIONAL SLIDE) Sufficient Statistics

Note that the ML estimates can also be written as

$$\hat{\Sigma} = \sum_n x_n x_n^T - \left(\sum_n x_n\right)\left(\sum_n x_n\right)^T, \quad \hat{\mu} = \frac{1}{N}\sum_n x_n$$

I.o.w., the two statistics (a 'statistic' is a function of the data) $\sum_n x_n$ and $\sum_n x_n x_n^T$ are sufficient to estimate the parameters μ and Σ from N observations. In the literature, $\sum_n x_n$ and $\sum_n x_n x_n^T$ are called **sufficient statistics**.

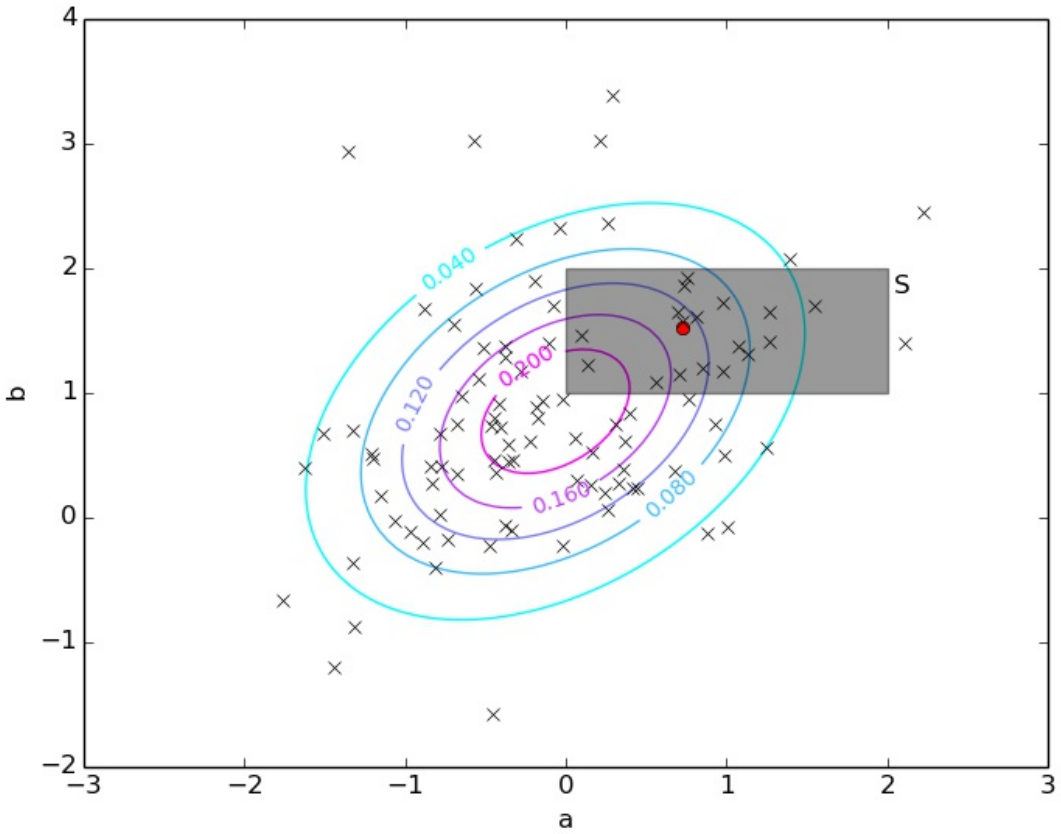
Sufficient statistics are useful because they summarize all there is to learn about the data set in a minimal set of variables .

Solution to Example Problem

We apply maximum likelihood estimation to fit a 2-dimensional Gaussian distribution m to data set D. Next, we evaluate $p(x_\bullet \in S|m)$ by (numerical) integration of the Gaussian pdf over S: $p(x_\bullet \in S|m) = \int_S p(x|m)\mathrm{d}x$.

```
using Cubature # Numerical integration package

# Maximum likelihood estimation of 2D Gaussian
μ = 1/N * sum(D,2)[:,1]
D_min_μ = D - repmat(μ, 1, N)
Σ = 1/N * D_min_μ*D_min_μ'
m = MvNormal(μ, Σ)

# Contour plot of estimated Gaussian density
A = Matrix{Float64}(100,100); B = Matrix{Float64}(100,100)
density = Matrix{Float64}(100,100)
for i=1:100
    for j=1:100
        A[i,j] = a = (i-1)*6/100.-2
        B[i,j] = b = (j-1)*6/100.-3
        density[i,j] = pdf(m, [a,b])
    end
end
c = contour(A, B, density, 6, zorder=1)
PyPlot.set_cmap("cool")
clabel(c, inline=1, fontsize=10)

# Plot observations, x•, and the countours of the estimated Gausian density
plotObservations(D)
plot(x_dot[1], x_dot[2], "ro")

# Numerical integration of p(x|m) over S:
(val,err) = hcubature((x)->pdf(m,x), [0., 1.], [2., 2.])
println("p(x∈S|m) ≈ $(val)")
```

```
p(x∈S|m) ≈ 0.1889431554757333
```



Discrete Data: the 1-of-K Coding Scheme

- Consider a coin-tossing experiment with outcomes $x \in \{0,1\}$ (corresponding to tail and head, resp.) and $0 \leq \mu \leq 1$ the probability of heads. This model can written as a **Bernoulli distribution**:

$$p(x|\mu) = \mu^x (1-\mu)^{1-x}$$

  - Note that in expression $\mu^x (1-\mu)^{1-x}$, the variable x acts as a (binary) **selector** for the tail or head probabilities. Think of this as an 'if'-statement in programming.

- **1-of-K scheme**. Now consider a K-sided coin, a.k.a. a *die* (pl.: dice). It will be very convenient to code the outcomes by a vector $x = (x_1, \ldots, x_K)^T$ with **binary selection variables**

$$x_k = \begin{cases} 1 & \text{if die landed on } k\text{th face} \\ 0 & \text{otherwise} \end{cases}$$

  - E.g., For $K = 6$, if the die lands on the 3rd face, we encode that as $x = (0,0,1,0,0,0)^T$.

- Assume the probabilities $p(x_k = 1) = \mu_k$ with $\sum_k \mu_k = 1$. The data generating distribution is then (note the similarity to the Bernoulli distribution)

$$p(x|\mu) = \mu_1^{x_1} \mu_2^{x_2} \cdots \mu_k^{x_k} = \prod_k \mu_k^{x_k}$$

- This generalized Bernoulli distribution is called the **categorical distribution** (or sometimes the 'multi-noulli' distribution).

- Note that $\sum_k x_k = 1$ and verify for yourself that $\mathbf{E}[x|\mu] = \mu$.

Categorical vs. Multinomial Distribution

- Observe a data set $D = \{x_1, \ldots, x_N\}$ of N IID rolls of a K-sided die, with generating PDF

$$p(D|\mu) = \prod_n \prod_k \mu_k^{x_{nk}} = \prod_k \mu_k^{\sum_n x_{nk}} = \prod_k \mu_k^{m_k}$$

  where $m_k = \sum_n x_{nk}$ is the total number of occurrences that we 'threw' k eyes.

- This distribution depends on the observations **only** through the quantities $\{m_k\}$, with generally $K \ll N$.

- A related distribution is the distribution over $D_m = \{m_1, \ldots, m_K\}$, which is called the **multinomial distribution**,

$$p(D_m|\mu) = \frac{N!}{m_1! m_2! \ldots m_K!} \prod_k \mu_k^{m_k}.$$

- $p(D|\mu) = p(\, x_1, \ldots, x_N \mid \mu\,)$ is a distribution over the individual observations, whereas $p(D_m|\mu) = p(\, m_1, \ldots, m_K \mid \mu\,)$ is a distribution over the data frequencies m_k.

Maximum Likelihood Estimation for the Multinomial

Now let's find the ML estimate for μ, based on N throws of a K-sided die. Again we use the shorthand $m_k \triangleq \sum_n x_{nk}$.

- The log-likelihood for the multinomial distribution is given by

$$\begin{aligned} \mathrm{L}(\mu) &\triangleq \log p(D_m|\mu) \propto \log \prod_k \mu_k^{m_k} \\ &= \sum_k m_k \log \mu_k \end{aligned} \tag{2}$$

- When doing ML estimation, we must obey the constraint $\sum_k \mu_k = 1$, which can be accomplished by a Lagrange multiplier. The augmented log-likelihood with Lagrange multiplier is then

$$\mathrm{L}'(\mu) = \sum_k m_k \log \mu_k + \lambda \cdot (1 - \sum_k \mu_k)$$

- Set derivative to zero yields the **sample proportion** for μ_k

$$\nabla_{\mu_k} \mathrm{L}' = \frac{m_k}{\hat{\mu}_k} - \lambda \stackrel{!}{=} 0 \;\Rightarrow\; \hat{\mu}_k = \frac{m_k}{N}$$

where we get λ from the constraint

$$\sum_k \hat{\mu}_k = \sum_k \frac{m_k}{\lambda} = \frac{N}{\lambda} \stackrel{!}{=} 1$$

Recap ML for Density Estimation

Given N IID observations $D = \{x_1, \ldots, x_N\}$

- For a **multivariate Gaussian** model $p(x_n|\theta) = \mathcal{N}(x_n|\mu, \Sigma)$, we obtain ML estimates

$$\hat{\mu} = \frac{1}{N} \sum_n x_n \qquad \text{(sample mean)}$$

$$\hat{\Sigma} = \frac{1}{N} \sum_n (x_n - \hat{\mu})(x_n - \hat{\mu})^T \qquad \text{(sample variance)}$$

- For discrete outcomes modeled by a 1-of-K **categorical distribution** we find

$$\hat{\mu}_k = \frac{1}{N} \sum_n x_{nk} \quad \left(= \frac{m_k}{N}\right) \qquad \text{(sample proportion)}$$

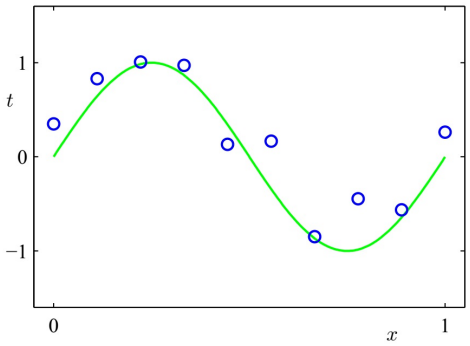
- Note the similarity for the means between discrete and continuous data.
- We didn't use a co-variance matrix for discrete data. Why?

Linear Regression

Preliminaries

- Goal
  - Maximum likelihood estimates for various linear regression variants
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 140-144

Regression – Illustration



Given a set of (noisy) data measurements, find the 'best' relation between an input variable x and input-dependent outcomes y.

The Regression Model

- Observe N IID data **pairs** $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ with $x_n \in \mathbb{R}^D$ and $y_n \in \mathbb{R}$.
- [Q.] We could try to build a model for the data by density estimation, $p(x, y)$, but what if we are interested only in (a model for) the responses y_n for **given inputs** x_n?
- [A.] We will build models for the conditional distribution $p(y|x)$.
  - Note that, since $p(x, y) = p(y|x)\, p(x)$, this is a building block for the joint data density.
- In a *regression model*, we try to 'explain the data' by a purely deterministic term $f(x, w)$, plus a purely random term ϵ_n for 'unexplained noise',

$$y_n = f(x_n, w) + \epsilon_n$$

Model Specification for Linear Regression

- In **linear** regression, we assume that
$$f(x, w) = w^T x\,.$$
- In **ordinary linear regression**, the noise process ϵ_n is zero-mean Gaussian with constant variance σ^2, i.e.
$$y_n = w^T x_n + \mathcal{N}(0, \sigma^2)\,,$$
or equivalently, the likelihood model is
$$p(y_n|\, x_n, w) = \mathcal{N}(y_n|\, w^T x_n, \sigma^2)\,.$$
- Remember that for full Bayesian learning we should also choose a prior $p(w)$; In ML estimation, the prior is uniformly distributed (so it can be ignored).

ML Estimation for Linear Regression Model

- **Inference** by ML; Let's first work out the log-likelihood for w,
$$\begin{aligned}\log p(D|w) &\overset{\text{IID}}{=} \sum_n \log \mathcal{N}(y_n|\, w^T x_n, \sigma^2) \\ &\propto -\frac{1}{2\sigma^2}\sum_n (y_n - w^T x_n)^2 \\ &= -\frac{1}{2\sigma^2}(y - \mathbf{X}w)^T (y - \mathbf{X}w)\end{aligned}$$
where we defined $N \times 1$ vector $y = (y_1, y_2, \ldots, y_N)^T$ and $(N \times D)$-dim matrix $\mathbf{X} = (x_1, x_2, \ldots, x_n)^T$.
- Set the derivative
$$\nabla_w \log p(D|w) = \frac{1}{\sigma^2}\mathbf{X}^T(y - \mathbf{X}w)$$
to zero for the maximum likelihood estimate
$$\hat{w}_{\text{ML}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T y$$
- The matrix $\mathbf{X}^\dagger \equiv (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ is also known as the **Moore-Penrose pseudo-inverse** (which is sort-of-an-inverse for non-square matrices).
- Note that size $(N \times D)$ of the data matrix $\mathbf{X}$ grows with number of observations, but the size $(D \times D)$ of $\mathbf{X}^T\mathbf{X}$ is independent of training data set.
- **Prediction** of new data points by
$$\hat{y}_{\text{new}} = \hat{w}_{\text{ML}}^T x_{\text{new}} = y^T \mathbf{X} \left(\mathbf{X}^T\mathbf{X}\right)^{-1} x_{\text{new}}$$
- Also note that the *prediction* of a new data point
$$\hat{y}_{\text{new}} = \hat{w}_{\text{ML}}^T x_{\text{new}} = x_{\text{new}}^T \hat{w}_{\text{ML}} = \left(x_{\text{new}}^T \mathbf{X}^\dagger\right) y$$
is a linear combination of the previous data points $y = (y_1, y_1, \ldots, y_N)^T$.

Deterministic Least-Squares Regression

- (You may say that) we don't need to work with probabilistic models. E.g., there's also the deterministic **least-squares** solution: minimize sum of squared errors,

$$\hat{w}_{LS} = \underset{w}{\operatorname{argmin}} \sum_n \left(y_n - w^T x_n\right)^2$$

- Setting the derivative

$$\partial (y - \mathbf{X}w)^T (y - \mathbf{X}w) / \partial w = -2\mathbf{X}^T (y - \mathbf{X}w)$$

to zero yields the **normal equations**

$$\mathbf{X}^T \mathbf{X} \hat{w}_{\mathrm{LS}} = \mathbf{X}^T y$$

and consequently

$$\hat{w}_{\mathrm{LS}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T y$$

$\Rightarrow$ Least-squares regression ($\hat{w}_{\mathrm{LS}}$) corresponds to (probabilistic) maximum likelihood ($\hat{w}_{\mathrm{ML}}$) if

1. **IID samples** (determines how errors are combined), and
2. Noise $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$ is **Gaussian** (determines error metric)

Probabilistic vs. Deterministic Approach

- The (deterministic) least-squares approach assumed IID Gaussian distributed data, but these assumptions are not obvious from looking at the least-squares (LS) criterion.
- If the data were better modeled by non-Gaussian assumptions (or not IID), then LS might not be appropriate.
- The probabilistic approach makes all these issues completely transparent by focusing on the **model specification** rather than the error criterion.
- Next, we will show this by two examples: (1) samples not identically distributed, and (2) few data points.

Not Identically Distributed Data

- What if we assume that the variance of the measurement error varies with the sampling index, $\epsilon_n \sim \mathcal{N}(0, \sigma_n^2)$?
- Let's make the log-likelihood again:

$$\begin{aligned} \mathrm{L(w)} &\triangleq \log p(D|w) \\ &\propto -\frac{1}{2} \sum_n \frac{(y_n - w^T x_n)^2}{\sigma_n^2} \\ &= -\frac{1}{2} (y - \mathbf{X}w)^T \Lambda (y - \mathbf{X}w) \end{aligned}$$

where $\Lambda \triangleq \mathrm{diag}[1/\sigma_n^2]$.

- Set derivative

$$\partial \mathrm{L(w)} / \partial w = -\mathbf{X}^T \Lambda (y - \mathbf{X}w)$$

to zero to get the **normal equations**

$$\mathbf{X}^T \Lambda \mathbf{X} \hat{w}_{\mathrm{WLS}} = \mathbf{X}^T \Lambda y$$

and consequently

$$\hat{w}_{\mathrm{WLS}} = \left(\mathbf{X}^T \Lambda \mathbf{X}\right)^{-1} \mathbf{X}^T \Lambda y$$

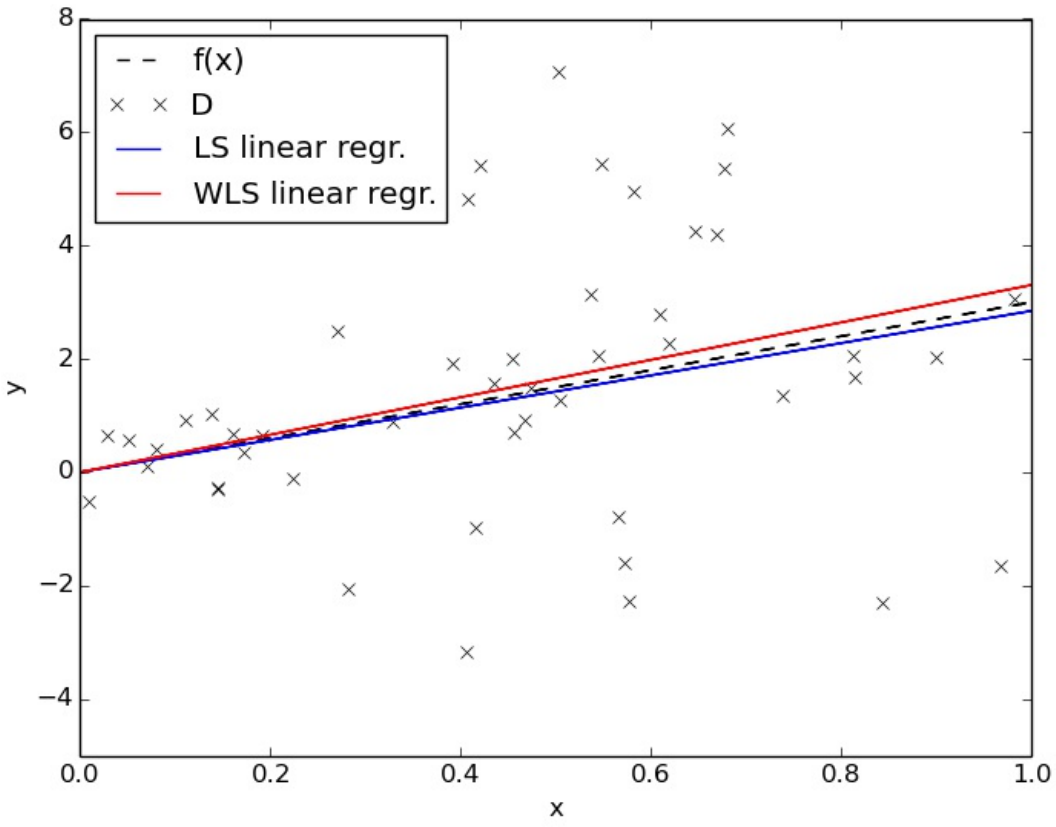
- This is called the **Weighted Least Squares** (WLS) solution. (Note that we just stumbled upon it, the crucial aspect is appropriate model specification!)
- Note also that the dimension of Λ grows with the number of data points. In general, models for which the number of parameters grow as the number of observations increase are called **non-parametric models**.

CODE EXAMPLE

We'll compare the Least Squares and Weighted Least Squares solutions for a simple linear regression model with input-dependent noise:

$$
\begin{aligned}
x &\sim \mathrm{Unif}[0,1] \\
y|x &\sim \mathcal{N}(f(x), v(x)) \\
f(x) &= 3x \\
v(x) &= 10e^{2x^2} - 9.5 \\
D &= \{(x_1, y_1), \ldots, (x_N, y_N)\}
\end{aligned}
$$

```
using PyPlot
# Model specification: y|x ~ (f(x), v(x))
f(X) = 3.*X
v(X) = exp(2*X.^2).*10 - 9.5 # Input dependent noise variance
X_test = [0.0, 1,0]
plot(X_test, f(X_test), "k--") # Plot f(x)

# Generate N samples (x,y), where x ~ Unif[0,1]
N = 50
X = rand(N)
y = f(X) + sqrt(v(X)) .* randn(N)
plot(X, y, "kx"); xlabel("x"); ylabel("y") # Plot samples

# LS regression
w_ls = inv(reshape(X'*X,1,1)) * X' * y # = pinv(X) * y
plot(X_test, w_ls.*X_test, "b-") # Plot LS solution

# Weighted LS regression
Λ = diagm(1./v(X))
w_wls = inv(reshape(X'*Λ*X,1,1)) * X' * Λ * y
plot(X_test, w_wls.*X_test, "r-") # Plot WLS solution
ylim([-5,8]); legend(["f(x)", "D", "LS linear regr.", "WLS linear regr."],loc=2);
```



END OF CODE EXAMPLE

Too Few Training Samples

- [Q.]: If we have fewer training samples than input dimensions, $\mathbf{X}^T\mathbf{X}$ will not be invertible.
- [A.]: In case of (expected) problems, **go back to full Bayesian!** Do proper model specification, Bayesian inference etc.
- **Model specification**. Let's try a Gaussian prior for w (why is this reasonable?),

$$p(w) = \mathcal{N}(w|0, \Sigma) = \mathcal{N}(w|0, \varepsilon I)$$

- **Learning**. Let's do Bayesian inference,

$$\begin{aligned}\log p(w|D) &\propto \log p(D|w)p(w) \\ &\overset{IID}{=} \log \sum_n p(y_n|x_n, w) + \log p(w) \\ &= \log \sum_n \mathcal{N}(y_n|\, w^T x_n, \sigma^2) + \log \mathcal{N}(w|0, \varepsilon I) \\ &\propto \frac{1}{2\sigma^2}(y - \mathbf{X}w)^T\,(y - \mathbf{X}w) + \frac{1}{2\epsilon} w^T w\end{aligned}$$

- **Done!** The posterior $p(w|D)$ specifies all we know about w after seeing the data D.
- As discussed, for practical purposes, you often want a point estimate for w, rather than a posterior distribution.
- For instance, let's take a **MAP estimate**. Set derivative

$$\nabla_w \log p(w|D) = -\frac{1}{\sigma^2}\mathbf{X}^T(y - \mathbf{X}w) + \frac{1}{\varepsilon} w$$

to zero, yielding

$$\hat{w}_{\text{MAP}} = \left(\mathbf{X}^T\mathbf{X} + \frac{\sigma^2}{\varepsilon} I\right)^{-1} \mathbf{X}^T y$$

- Note that, in contrast to $\mathbf{X}^T\mathbf{X}$, the matrix $\left(\mathbf{X}^T\mathbf{X} + (\sigma^2/\varepsilon)I\right)$ is always invertible! Why?
- Note also that $\hat{w}_{\text{LS}}$ is retrieved by letting $\varepsilon \to \infty$. Does that make sense?

Adaptive Linear Regression

- What if the data arrives one point at a time?
- Two standard *adaptive* linear regression approaches: RLS and LMS. Here we shortly recap the LMS approach.
- **Least Mean Squares** (LMS) is gradient-descent on a 'local-in-time' approximation of the square-error cost function.
- Define the cost-of-current-sample as

$$E_n(w) = \frac{1}{2}(y_n - w^T x_n)^2$$

and track the optimum by gradient descent (at each sample index n):

$$w_{n+1} = w_n - \eta \left.\frac{\partial E_n}{\partial w}\right|_{w_n}$$

which leads to the LMS update:

$$w_{n+1} = w_n + \eta\,(y_n - w_n^T x_n)\,x_n$$

- (OPTIONAL) Is there also a Bayesian treatment of LMS? Sure, e.g., have a look at G. Deng et al., *A model-based approach for the development of LMS algorithms*', ISCAS-05 symposium, 2005.

Generative Classification

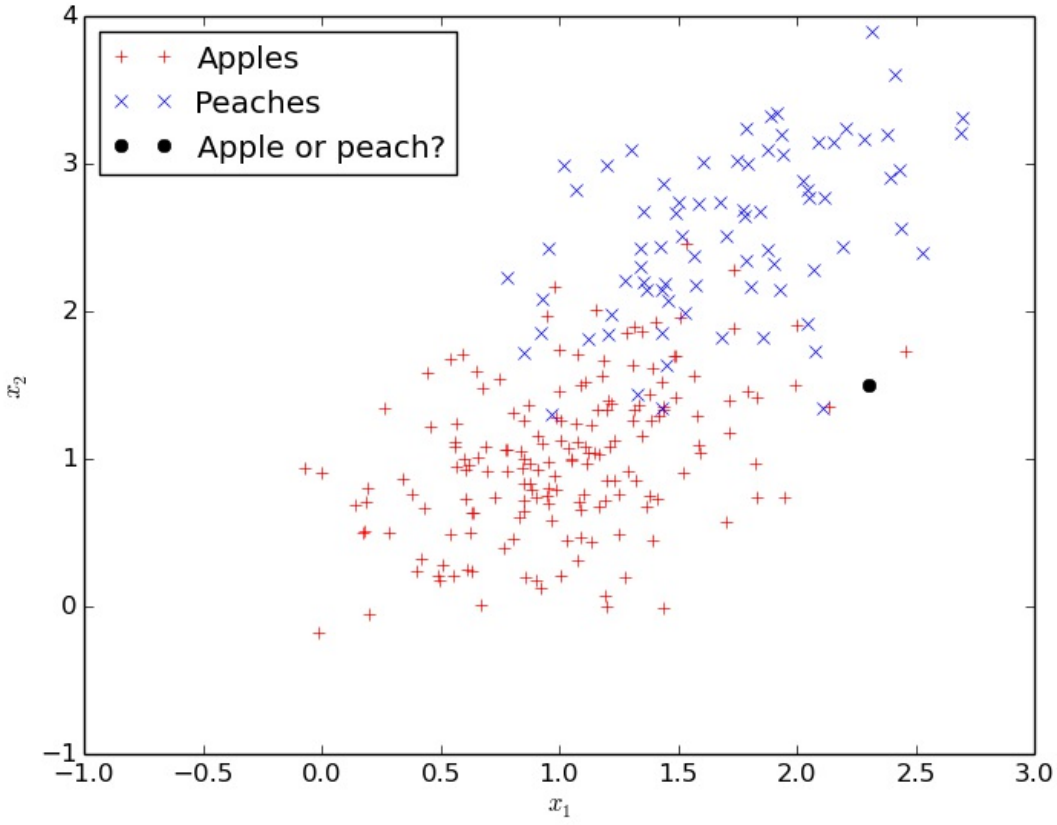
Preliminaries

- Goal
  - Introduction to linear generative classification with multinomial-Gaussian generative model
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 196-202

Example Problem: an apple or a peach?

You're given numerical values for the skin features _roughness_ and _color_ for 200 pieces of fruit, where for each piece of fruit you also know if it is an apple or a peach. Now you receive the roughness and color values for a new piece of fruit but you don't get its class label (apple or peach). What is the probability that the new piece is an apple?

```
using Distributions, PyPlot
N = 250; p_apple = 0.7; Σ = [0.2 0.1; 0.1 0.3]
p_given_apple = MvNormal([1.0, 1.0], Σ) # p(X|y=apple)
p_given_peach = MvNormal([1.7, 2.5], Σ) # p(X|y=peach)
X = Matrix{Float64}(2,N); y = Vector{Bool}(N) # true corresponds to apple
for n=1:N
    y[n] = (rand() < p_apple) # Apple or peach?
    X[:,n] = y[n] ? rand(p_given_apple) : rand(p_given_peach) # Sample features
end
X_apples = X[:,find(y)]'; X_peaches = X[:,find(!y)]' # Sort features on class
x_test = [2.3; 1.5] # Features of 'new' data point

function plot_fruit_dataset()
    # Plot the data set and x_test
    plot(X_apples[:,1], X_apples[:,2], "r+")   # apples
    plot(X_peaches[:,1], X_peaches[:,2], "bx") # peaches
    plot(x_test[1], x_test[2], "ko")           # 'new' unlabelled data point
    legend(["Apples"; "Peaches"; "Apple or peach?"], loc=2)
    xlabel(L"x_1"); ylabel(L"x_2"); xlim([-1,3]); ylim([-1,4])
end
plot_fruit_dataset();
```



Generative Classification Problem Statement

- Given is a data set $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
  - inputs $x_n \in \mathbb{R}^D$ are called **features**.
  - outputs $y_n \in \mathcal{C}_k$, with $k = 1, \ldots, K$; The **discrete** targets $\mathcal{C}_k$ are called **classes**.
- We will again use the 1-of-K notation for the discrete classes. Define the binary **class selection variable**

$$y_{nk} = \begin{cases} 1 & \text{if } y_n \text{ in class } \mathcal{C}_k \\ 0 & \text{otherwise} \end{cases}$$

- The plan for generative classification: build a model for the joint pdf $p(x, y) = p(x|y)p(y)$ and use Bayes to infer the posterior class probabilities

$$p(y|x) = \frac{p(x|y)p(y)}{\sum_y p(x|y)p(y)}$$

1 – Model specification

Likelihood

- Assume Gaussian **class-conditional distributions** with **constant covariance matrix** across the classes,
$$p(x_n|\mathcal{C}_k) = \mathcal{N}(x_n|\mu_k, \Sigma)$$
with notational shorthand: $\mathcal{C}_k \triangleq (y_{nk} = 1)$.

Prior

- We use a categorical distribution for the class labels y_{nk}:
$$p(\mathcal{C}_k) = \pi_k$$
- This leads to
$$p(x_n, \mathcal{C}_k) = \pi_k \cdot \mathcal{N}(x_n|\mu_k, \Sigma)$$
- The log-likelihood for the full data set is then
$$\begin{aligned}
\log p(D|\theta) &\overset{\text{IID}}{=} \sum_n \log p(x_n, \mathcal{C}_1, \ldots, \mathcal{C}_K \,|\, \theta) \\
&= \sum_n \log \prod_k p(x_n, \mathcal{C}_k|\theta)^{y_{nk}} \quad \text{(use 1-of-K coding)} \\
&= \sum_{n,k} y_{nk} \log p(x_n, \mathcal{C}_k|\theta) \\
&= \sum_{n,k} y_{nk} \log \mathcal{N}(x_n|\mu_k, \Sigma) + \sum_{n,k} y_{nk} \log \pi_k \\
&= \sum_{n,k} y_{nk} \underbrace{\log \mathcal{N}(x_n|\mu_k, \Sigma)}_{\text{Gaussian}} + \underbrace{\sum_k m_k \log \pi_k}_{\text{multinomial}}
\end{aligned}$$
where we used $m_k \triangleq \sum_n y_{nk}$.
- As usual, the rest (inference for parameters and model prediction) through straight probability theory.

2 – Parameter Inference for Classification

We'll do ML estimation of $\theta = \{\pi_k, \mu_k, \Sigma\}$ from data D

- Recall (from the previous slide) the log-likelihood

$$\log p(D|\theta) = \sum_{n,k} y_{nk} \underbrace{\log \mathcal{N}(x_n|\mu_k, \Sigma)}_{\text{Gaussian}} + \underbrace{\sum_k m_k \log \pi_k}_{\text{multinomial}}$$

- Maximization of the LLH breaks down into
  - **Gaussian density estimation** for parameters μ_k, Σ, since the first term contains exactly the LLH for MVG density estimation (see lesson on Density Est., Eq.1)
  - **Multinomial density estimation** for class priors π_k, since the second term holds exactly the LLH for multinomial density estimation (see lesson on Density Estimation, Eq.2).

It follows (by similar computations) that

- ML for multinomial prior (we've done this before!)

$$\hat{\pi}_k = m_k / N \qquad \text{(class prior)}$$

- Now group the data into separate classes and do MVG ML estimation for class-conditional parameters (we've done this as well).

$$\hat{\mu}_k = \frac{\sum_n y_{nk} x_n}{\sum_n y_{nk}} = \frac{1}{m_k} \sum_n y_{nk} x_n \qquad \text{(class-cond. mean)}$$

$$\begin{aligned} \hat{\Sigma} &= \frac{1}{N} \sum_{n,k} y_{nk} (x_n - \hat{\mu}_k)(x_n - \hat{\mu}_k)^T \qquad \text{(variance)} \\ &= \sum_k \hat{\pi}_k \cdot \underbrace{\left(\frac{1}{m_k} \sum_n y_{nk} (x_n - \hat{\mu}_k)(x_n - \hat{\mu}_k)^T \right)}_{\text{class-cond. variance}} \\ &= \sum_k \hat{\pi}_k \cdot \hat{\Sigma}_k \end{aligned}$$

- Note that y_{nk} groups data from the same class.

3 – Application: Class prediction for new Data

- Given a 'new' input $x_\bullet$, use Bayes rule to get posterior class probability

$$\begin{aligned} p(\mathcal{C}_k|x_\bullet, \theta) &\propto p(\mathcal{C}_k)\, p(x_\bullet|\mathcal{C}_k) \\ &\propto \pi_k \exp\left\{ -\frac{1}{2}(x_\bullet - \mu_k)^T \Sigma^{-1} (x_\bullet - \mu_k) \right\} \\ &\propto \exp\left\{ \mu_k^T \Sigma^{-1} x_\bullet - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k \right\} \\ &= \exp\{\beta_k^T x + \gamma_k\} \end{aligned}$$

  where

$$\begin{aligned} \beta_k &= \Sigma^{-1} \mu_k \\ \gamma_k &= -\frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k \,. \end{aligned}$$

- The class posterior $\frac{\exp(a_k)}{\sum_{k'} \exp(a_{k'})}$ is called a **softmax** function.

Discrimination Boundaries

- The class posterior $\log p(\mathcal{C}_k|x) \propto \beta_k^T x + \gamma_k$ is a linear function of the input features.
- Thus, the contours of equal probability (**discriminant functions**) are lines (hyperplanes) in feature space"

$$\log \frac{p(\mathcal{C}_k|x,\theta)}{p(\mathcal{C}_j|x,\theta)} = \beta_{kj}^T x + \gamma_{kj} = 0$$

  where we defined $\beta_{kj} \triangleq \beta_k - \beta_j$ and similarly for γ_{kj}.
- (homework). What happens if we had not assumed class-independent variances $\Sigma_k = \Sigma$? Are the discrimination functions still linear? quadratic?
- How to classify a new input $x_\bullet$? The Bayesian answer is a posterior distribution $p(\mathcal{C}_k|x_\bullet)$. If you must choose, then the class with maximum posterior class probability

$$\begin{aligned} k^* &= \operatorname{argmax}_k p(\mathcal{C}_k|x_\bullet) \\ &= \operatorname{argmax}_k \left(\beta_k^T x_\bullet + \gamma_k\right) \end{aligned}$$

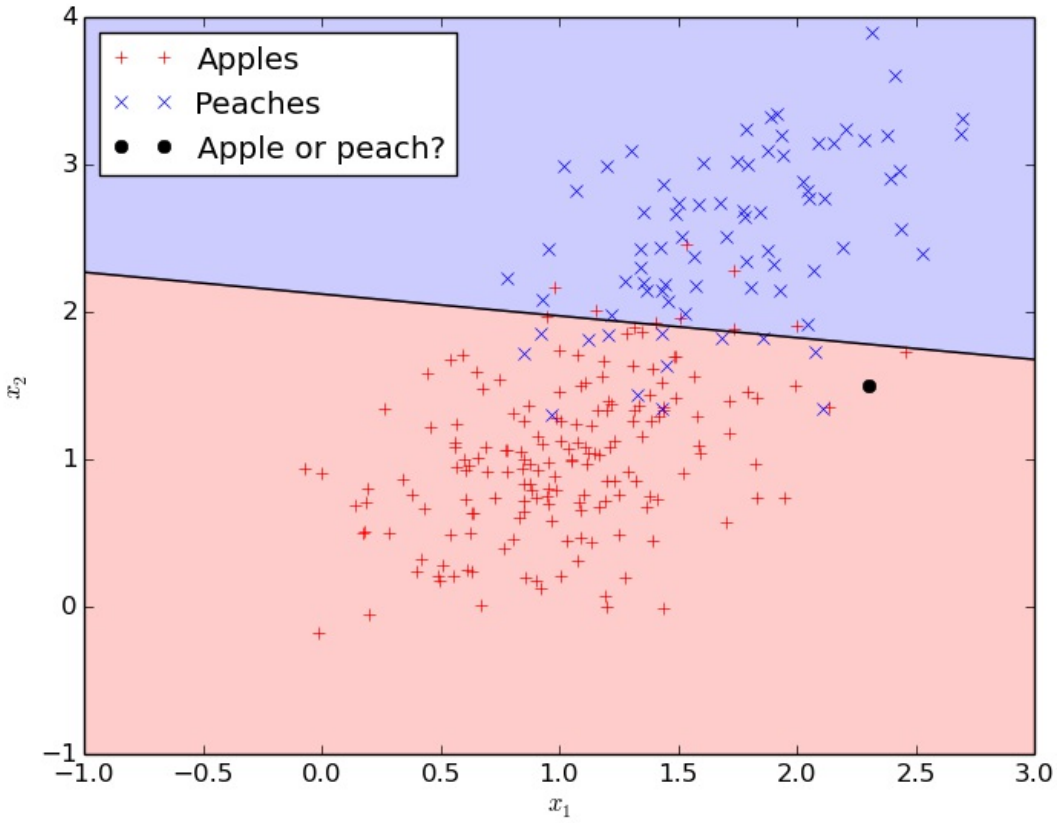
  is an appealing decision.

CODE EXAMPLE

We'll apply the above results to solve the "apple or peach" example problem.

```
# Make sure you run the data-generating code cell first

# Multinomial (in this case binomial) density estimation
p_apple_est = sum(y.==true) / length(y)
π_hat = [p_apple_est; 1-p_apple_est]

# Estimate class-conditional multivariate Gaussian densities
d1 = fit_mle(FullNormal, X_apples')  # MLE density estimation d1 = N(μ₁, Σ₁)
d2 = fit_mle(FullNormal, X_peaches') # MLE density estimation d2 = N(μ₂, Σ₂)
Σ = π_hat[1]*cov(d1) + π_hat[2]*cov(d2) # Combine Σ₁ and Σ₂ into Σ
conditionals = [MvNormal(mean(d1), Σ); MvNormal(mean(d2), Σ)] # p(x/C)

# Calculate posterior class probability of x• (prediction)
function predict_class(k, X) # calculate p(Ck|X)
    norm = π_hat[1]*pdf(conditionals[1],X) + π_hat[2]*pdf(conditionals[2],X)
    return π_hat[k]*pdf(conditionals[k], X) ./ norm
end
println("p(apple|x=x•) = $(predict_class(1,x_test))")

# Discrimination boundary of the posterior (p(apple|x;D) = p(peach|x;D) = 0.5)
β(k) = inv(Σ)*mean(conditionals[k])
γ(k) = -0.5 * mean(conditionals[k])' * inv(Σ) * mean(conditionals[k]) + log(π_hat[k])
function discriminant_x2(x1)
    # Solve discriminant equation for x2
    β12 = β(1) - β(2)
    γ12 = (γ(1) - γ(2))[1,1]
    return -1*(β12[1]*x1 + γ12) ./ β12[2]
end
plot_fruit_dataset() # Plot dataset
x1 = linspace(-1,3,10)
plot(x1, discriminant_x2(x1), "k-") # Plot discrimination boundary
fill_between(x1, -1, discriminant_x2(x1), color="r", alpha=0.2)
fill_between(x1, discriminant_x2(x1), 4, color="b", alpha=0.2);
```

```
p(apple|x=x•) = 0.8100962204940567
```



END OF CODE EXAMPLE

Recap Generative Classification

- Model spec. $p(x, \mathcal{C}_k \,|\, \theta) = \pi_k \cdot \mathcal{N}(x|\mu_k, \Sigma)$
- If the class-conditional distributions are Gaussian with equal covariance matrices across classes ($\Sigma_k = \Sigma$), then the discriminant functions are hyperplanes in feature space.
- ML estimation for $\{\pi_k, \mu_k, \Sigma\}$ breaks down to simple density estimation for Gaussian and multinomial.
- Posterior class probability is a softmax function

$$p(\mathcal{C}_k|x, \theta) \propto \exp\{\beta_k^T x + \gamma_k\}$$

where $\beta_k = \Sigma^{-1}\mu_k$ and $\gamma_k = -\frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log \pi_k$.

Discriminative Classification

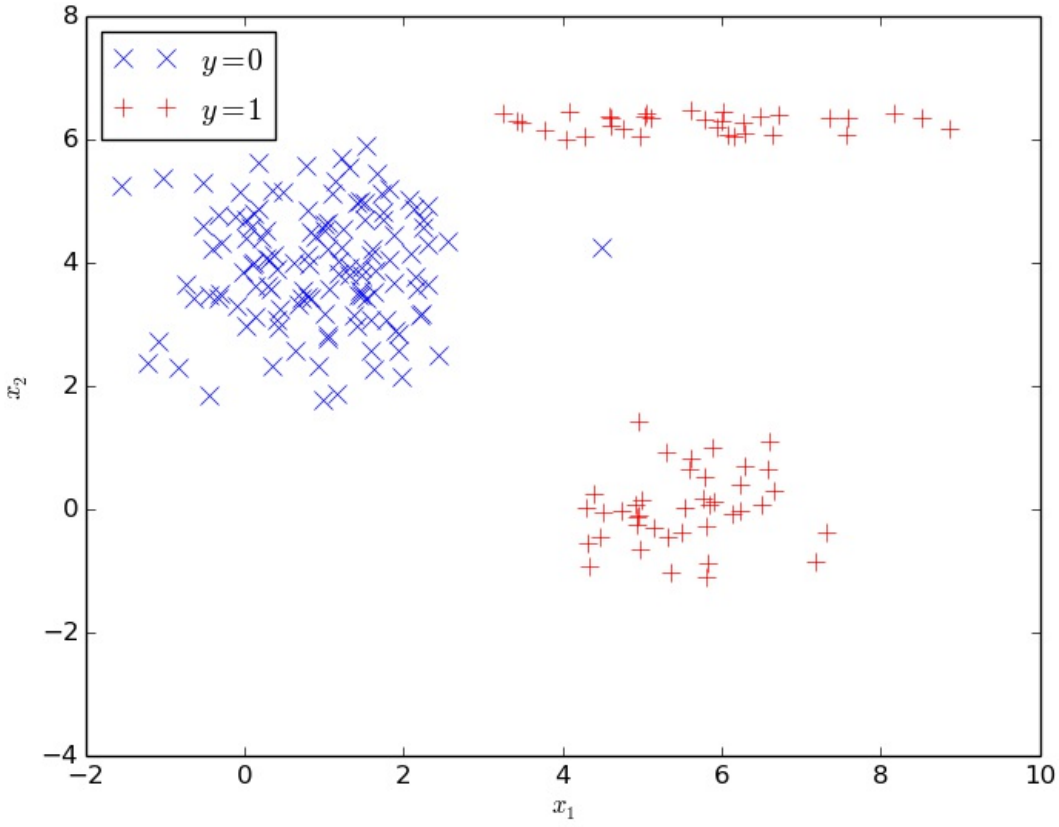
Preliminaries

- Goal
  - Introduction to discriminative classification models
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 203-206
    - T. Minka (2005), Discriminative models, not discriminative training

Problem: difficult class-conditional data distribitions

Our task will be the same as in the preceding class on (generative) classification. But this time, the class-conditional data distributions look very non-Gaussian, yet the linear discriminative boundary looks easy enough:

```
# Generate dataset {(x1,y1),...,(xN,yN)}
# x is a 2-d feature vector [x_1;x_2]
# y ∈ {false,true} is a binary class label
# p(x|y) is multi-modal (mixture of uniform and Gaussian distributions)
using PyPlot
include("scripts/lesson8_helpers.jl")
N = 200
X, y = genDataset(N) # Generate data set, collect in matrix X and vector y
X_c1 = X[:,find(!y)]'; X_c2 = X[:,find(y)]' # Split X based on class label
function plotDataSet()
    plot(X_c1[:,1], X_c1[:,2], "bx", markersize=8)
    plot(X_c2[:,1], X_c2[:,2], "r+", markersize=8, fillstyle="none")
    xlabel(L"x_1"); ylabel(L"x_2"); legend([L"y=0", L"y=1"], loc=2)
    xlim([-2;10]); ylim([-4, 8])
end
plotDataSet();
```



In this class, we will build a classifier for such cases.

Main Idea of Discriminative Classification

- Again, a data set is given by $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ with $x_n \in \mathbb{R}^D$ and $y_n \in \mathcal{C}_k$, with $k = 1, \ldots, K$.
- Sometimes, the precise assumptions of the (multinomial-Gaussian) generative model
$$p(x_n, \mathcal{C}_k|\theta) = \pi_k \cdot \mathcal{N}(x_n|\mu_k, \Sigma)$$
clearly do not match the data distribution.

Idea

- Model the posterior
$$p(\mathcal{C}_k|x_n)$$
directly, without any assumptions on the class densities.
- Of course, this implies also that we build direct models for the **discrimination boundaries**
$$\log \frac{p(\mathcal{C}_k|x_n)}{p(\mathcal{C}_j|x_n)} \stackrel{!}{=} 0$$

Model Specification

- [Q.]What model should we use for $p(\mathcal{C}_k|x_n)$?
- [A.] Get inspiration from the generative approach: choose the familiar softmax structure for the posterior class probability

$$p(\mathcal{C}_k|x_n,\theta) = \frac{e^{\theta_k^T x_n}}{\sum_j e^{\theta_j^T x_n}}$$

but **do not impose a Gaussian structure on the classes**.

⇒ There are **two key differences** between the discriminative and generative approach:

1. In the discriminative approach, the parameters θ_k are **not** structured into $\{\mu_k, \Sigma, \pi_k\}$. This provides discriminative approach with more flexibility.
2. ML learning for the discriminative approach by optimization of *conditional* likelihood $\prod_n p(y_n|x_n,\theta)$ rather than *joint* likelihood $\prod_n p(y_n,x_n|\theta)$.

ML Estimation for Discriminative Classification

- As we will see, ML estimation for discriminative classification is more complex than for the linear Gaussian generative approach.
- Work out the conditional log-likelihood

$$\mathrm{L}(\theta) = \log \prod_n \prod_k \underbrace{p(\mathcal{C}_k|x_n,\theta)}_{p_{nk}}^{y_{nk}} = \sum_{n,k} y_{nk} \log p_{nk}$$

- Use the fact that the softmax $\phi_k = e^{z_k}/\sum_j e^{z_j}$ has analytical derivative:

$$\begin{aligned}\frac{\partial \phi_k}{\partial z_j} &= \frac{(\sum_j e^{z_j}) e^{z_k}\delta_{kj} - e^{z_j}e^{z_k}}{(\sum_j e^{z_j})^2} = \frac{e^{z_k}}{\sum_j e^{z_j}}\delta_{kj} - \frac{e^{z_j}}{\sum_j e^{z_j}}\frac{e^{z_k}}{\sum_j e^{z_j}} \\ &= \phi_k \cdot (\delta_{kj} - \phi_j)\end{aligned}$$

- Take the derivative of $\mathrm{L}(\theta)$ (or: how to spend a hour ...)

$$\begin{aligned}\nabla_{\theta_j}\mathrm{L}(\theta) &= \sum_{n,k} \frac{\partial \mathrm{L}_{nk}}{\partial p_{nk}} \cdot \frac{\partial p_{nk}}{\partial z_{nj}} \cdot \frac{\partial z_{nj}}{\partial \theta_j} \\ &= \sum_{n,k} \frac{y_{nk}}{p_{nk}} \cdot p_{nk}(\delta_{kj} - p_{nj}) \cdot x_n \\ &= \sum_n \Big(y_{nj}(1-p_{nj}) - \sum_{k\neq j} y_{nk} p_{nj} \Big) \cdot x_n \\ &= \sum_n (y_{nj} - p_{nj}) \cdot x_n \\ &= \sum_n \Big(\underbrace{y_{nj}}_{\text{target}} - \underbrace{\frac{e^{\theta_j^T x_n}}{\sum_{j'} e^{\theta_{j'}^T x_n}}}_{\text{prediction}} \Big) \cdot x_n\end{aligned}$$

- The derivation for the derivative was painful, but the result is extremely simple. Compare the gradients for linear and logistic regression:

$$\nabla_\theta \mathrm{L}(\theta) = \sum_n \left(y_n - \theta^T x_n\right) x_n \qquad \text{(linear)}$$

$$\nabla_\theta \mathrm{L}(\theta) = \sum_n \left(y_n - \frac{1}{1+e^{-\theta^T x_n}}\right) x_n \qquad \text{(logistic)}$$

In both cases

$$\nabla_\theta \mathrm{L} = \sum_n (\mathrm{target}_n - \mathrm{prediction}_n) \cdot \mathrm{input}_n$$

- The parameter vector θ for logistic regression can be estimated through iterative gradient-based adaptation. E.g. (with iteration index i),

$$\hat{\theta}^{(i+1)} = \hat{\theta}^{(i)} + \eta \cdot \nabla_\theta \mathrm{L}(\theta)|_{\theta=\hat{\theta}^{(i)}}$$

Application – Classify a new input

- Discriminative model-based prediction for a new input $x_\bullet$ is easy, namely substitute the ML estimate in the model to get

$$p(\mathcal{C}_k|\, x_\bullet, \hat{\theta}) = \frac{\exp\left(\hat{\theta}_k^T x_\bullet\right)}{\sum_{k'} \exp\left(\hat{\theta}_{k'}^T x_\bullet\right)} \propto \exp\left(\hat{\theta}_k^T x_\bullet\right)$$

- The contours of equal probability (**discriminant boundaries**) are lines (hyperplanes) in feature space given by

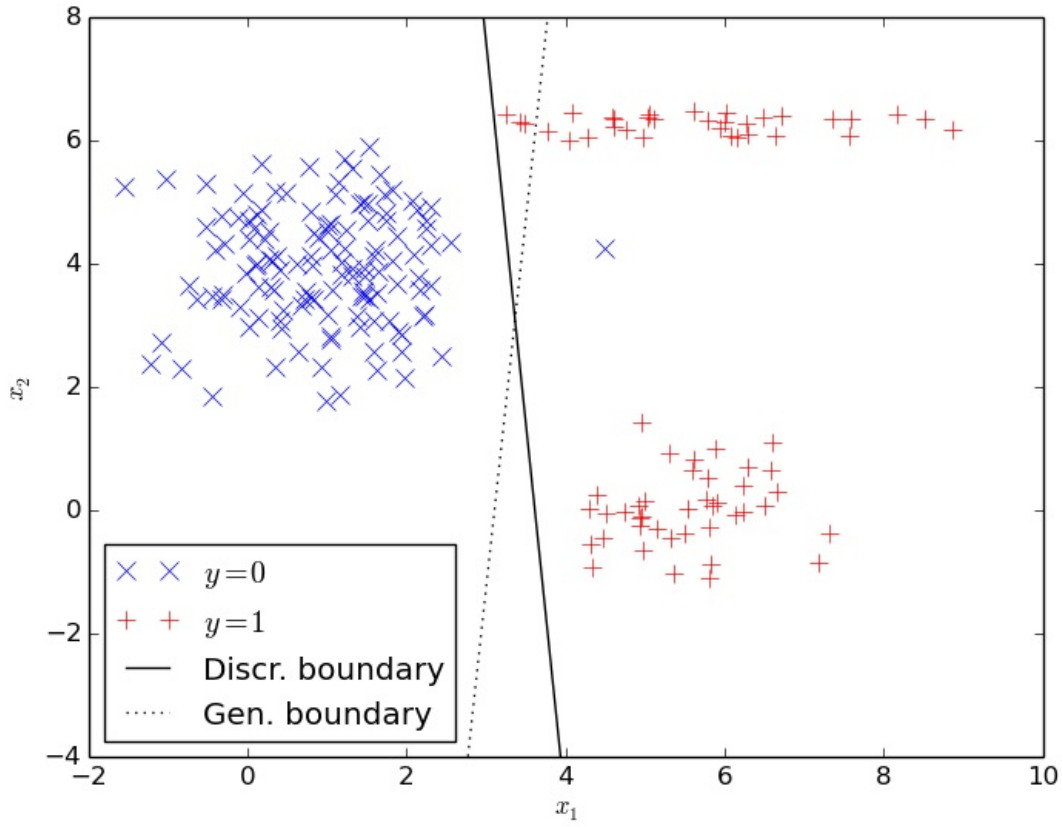
$$\log \frac{p(\mathcal{C}_k|x, \hat{\theta})}{p(\mathcal{C}_j|x, \hat{\theta})} = \left(\hat{\theta}_k - \hat{\theta}_j\right)^T x = 0$$

CODE EXAMPLE

Let us perform ML estimation of θ on the data set from the introduction. To allow an offset in the discrimination boundary, we add a constant 1 to the feature vector x. We only have to specify the (negative) log-likelihood and the gradient w.r.t. θ. Then, we use an off-the-shelf optimisation library to minimize the negative log-likelihood.

We plot the resulting maximum likelihood discrimination boundary. For comparison we also plot the ML discrimination boundary obtained from the generative Gaussian classifier from lesson 7.

```
using Optim # Optimization library

y_1 = zeros(length(y)) # class 1 indicator vector
y_1[y.==false] = 1.
X_ext = vcat(X, ones(1, length(y))) # Extend X with a row of ones to allow an offset in the discri
mination boundary

# Implement negative log-likelihood function
function neg_loglik_and_gradient!(θ::Vector, gradient::Vector)
    # Return negative log-likelihood: -L(θ)
    # Write gradient to negative log-likelihood to gradient argument
    p_1 = 1.0 ./ (1.0 + exp(-1.*(X_ext' * θ)[:,1]))   # P(C1|X,θ)
    gradient[:] = -1. .* X_ext * (y_1-p_1)

    return -1.0 * sum(log( (y_1.*p_1) + ((1.-y_1).*(1.-p_1))) ) # negative log-likelihood
end
neg_loglik(θ) = neg_loglik_and_gradient!(θ, zeros(length(θ)))

# Use Optim.jl optimiser to minimize the negative log-likelihood function w.r.t. θ
objective = DifferentiableFunction(neg_loglik, neg_loglik_and_gradient!, neg_loglik_and_gradient!)
results = optimize(objective, [0.0; 0.0; 0.0])
θ = results.minimum

# Plot the data set and ML discrimination boundary
plotDataSet()
p_1(x) = 1.0 ./ (1.0 + exp(-1.*([x;1.]' * θ)[:,1]))
boundary(x1) = -1./θ[2] * (θ[1]*x1 + θ[3])
plot([-2.;10.], boundary([-2.;10.]), "k-");

# Also fit the generative Gaussian model from lesson 7 and plot the resulting discrimination bound
ary for comparison
generative_boundary = buildGenerativeDiscriminationBoundary(X, y)
plot([-2.;10.], generative_boundary([-2.;10.]), "k:");
legend([L"y=0";L"y=1";"Discr. boundary";"Gen. boundary"], loc=3);
```



Given $\hat{\theta}$, we can classify a new input $x_\bullet = [4.0, 2.0]^T$:

```
x_test = [4.0; 2.0]
println("P(C1|x•,θ) = $(p_1(x_test))")
```

```
P(C1|x•,θ) = [0.11663485521519054]
```

The generative model gives a bad result because the feature distribution of one class is clearly non-Gaussian: the model does not fit the data well. The discriminative approach does not suffer from this problem because it makes no assumptions about the feature distribition $p(x|y)$, it just estimates the conditional class distribution $p(y|x)$ directly.

END OF CODE EXAMPLE

Recap Classification

	Generative	Discriminative
1	Like **density estimation**, model joint prob. $p(\mathcal{C}_k)p(x\|\mathcal{C}_k) = \pi_k \mathcal{N}(\mu_k, \Sigma)$	Like (linear) **regression**, model conditional $p(\mathcal{C}_k\|x, \theta)$
2	Leads to **softmax** posterior class probability $p(\mathcal{C}_k\|x, \theta) = e^{\theta_k^T x}/Z$ with **structured** θ	**Choose** also softmax posterior class probability $p(\mathcal{C}_k\|x, \theta) = e^{\theta_k^T x}/Z$ but now with 'free' θ
3	For Gaussian $p(x\|\mathcal{C}_k)$ and multinomial priors, $\hat{\theta}_k = \begin{bmatrix} -\frac{1}{2}\mu_k^T \sigma^{-1} \mu_k + \log \pi_k \\ \sigma^{-1}\mu_k \end{bmatrix}$ **in one shot**.	Find $\hat{\theta}_k$ through gradient-based adaptation $\nabla_{\theta_k} \mathrm{L}(\theta) = \sum_n \left(y_{nk} - \frac{e^{\theta_k^T x_n}}{\sum_{k'} e^{\theta_{k'}^T x_n}} \right) x_n$

Clustering with Gaussian Mixture Models

Preliminaries

- Goal
  - Introduction to the Expectation-Maximization (EM) Algorithm with application to Gaussian Mixture Models (GMM)
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 430-439 for Gaussian Mixture Models

Limitations of Simple IID Gaussian Models

Sofar, model inference was solved analytically, but we used strong assumptions:

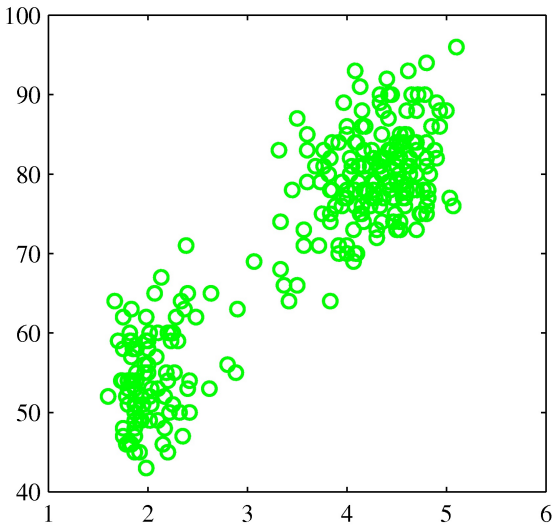
- IID sampling, $p(D) = \prod_n p(x_n)$
- Simple Gaussian (or multinomial) PDFs, $p(x_n) \sim \mathcal{N}(x_n|\mu, \Sigma)$
- Some limitations of Simple Gaussian Models with IID Sampling:
  1. What if the PDF is **multi-modal** (or is just not Gaussian in any other way)?
  2. Covariance matrix σ has $D(D+1)/2$ parameters.
     - This quickly becomes **a very large number** for increasing dimension D.
  3. Temporal signals are often **not IID**.

Towards More Flexible Models

- What if the PDF is multi-modal (or is just not Gaussian in any other way)?
  - **Discrete latent** variable models (a.k.a. **mixture** models). We'll cover this case in this lesson .
- Covariance matrix Σ has $D(D+1)/2$ parameters. This quickly becomes very large for increasing dimension D.
  - **Continuous latent** variable models (a.k.a. **dimensionality reduction** models). Covered in lesson 11.
- Temporal signals are often not IID.
  - Introduce **Markov dependencies** and **latent state** variable models. This will be covered in lesson 13.

Illustrative Example

- You're now asked to build a density model for a data set (Old Faithful, Bishop pg. 681) that clearly is not distributed as a single Gaussian:



Unobserved Classes

Consider again a set of observed data $D = \{x_1, \ldots, x_N\}$

- This time we suspect that there are unobserved class labels that would help explain (or predict) the data, e.g.,
  - the observed data are the color of living things; the unobserved classes are animals and plants.
  - observed are wheel sizes; unobserved categories are trucks and personal cars.
  - observed is an audio signal; unobserved classes include speech, music, traffic noise, etc.
- Classification problems with unobserved classes are called **Clustering** problems. The learning algorithm needs to **discover the classes from the observed data**.

Latent Variable Model Specification

- If the categories were observed as well, these data could be nicely modeled by the previously discussed generative classification framework.
- Let us use **equivalent model assumptions to linear generative classification**

$$p(x_n, \mathcal{C}_k) = p(\mathcal{C}_k)\, p(x_n|\mathcal{C}_k) = \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)$$

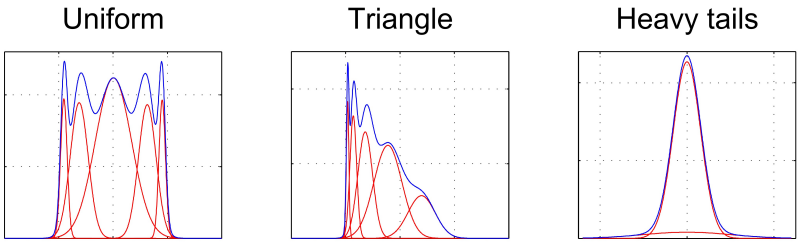
- Since the class labels are not observed, the probability for observations is obtained through *marginalization* over the classes (this is different in classification problems, where the classes are observed):

$$p(x_n) = \sum_k \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)$$

  - The class priors $p(\mathcal{C}_k) = \pi_k$ are often called *mixture coefficients*.
- This model is called a **Gaussian Mixture Model**.

Gaussian Mixture Models

- GMMs are very popular models. They have decent computational properties and are **universal approximators of densities** (as long as there are enough Gaussians of course)



Inference: Log-Likelihood for GMM

- The log-likelihood for observed data $D = \{x_1, \ldots, x_N\}$,

$$\begin{aligned}\log p(D|\theta) &\overset{\text{IID}}{=} \sum_n \log p(x_n|\theta) \\ &= \sum_n \log \left(\sum_{k=1}^{K} p(\mathcal{C}_k)\, p(x_n|\mathcal{C}_k) \right) \\ &= \sum_n \log \left(\sum_k \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k) \right)\end{aligned}$$

  ... and now the log-of-sum cannot be further simplified.

- Compare this to the log-likelihood for $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ in (generative) classification:

$$\begin{aligned}\log p(D|\theta) &= \sum_n \log \prod_k p(x_n, \mathcal{C}_k|\theta)^{y_{nk}} \\ &= \sum_{n,k} y_{nk} \log p(x_n, \mathcal{C}_k|\theta) \\ &= \sum_k m_k \log \pi_k + \sum_{n,k} y_{nk} \log \mathcal{N}(x_n|\mu_k, \Sigma)\end{aligned}$$

  which led to easy Gaussian and multinomial parameter estimation.

- Fortunately GMMs can be trained by maximum likelihood using an efficient algorithm: Expectation-Maximization.

The Posterior Class Probability as a Soft Class Indicator

- We don't *observe* the class labels, but we can compute the posterior probability for the class labels, given the observation x_n:

$$\begin{aligned}\gamma_{nk} &\triangleq p(\mathcal{C}_k|x_n) \\ &= \frac{p(x_n|\mathcal{C}_k)p(\mathcal{C}_k)}{\sum_{k'} p(x_n|\mathcal{C}_{k'})p(\mathcal{C}_{k'})} \\ &= \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{k'} \pi_{k'} \mathcal{N}(x_n|\mu_{k'}, \Sigma_{k'})}\end{aligned}$$

- Note that $0 \leq \gamma_{nk} \leq 1$ and is available (i.e., can be evaluated).

- γ_{nk} are soft class indicators, a.k.a. **reponsibilities**. Responsibility variables play the same role in clustering as class selection variables (y_{nk}) do in classification problems.

- For more elaborate models, it is useful to associate a *hidden* 1-of-K selector variable variable z_{nk} with each observation x_n as

$$z_{nk} = \begin{cases} 1 & \text{if } z_n \text{ in class } \mathcal{C}_k \\ 0 & \text{otherwise} \end{cases}$$

- The responsibility is then given by $\gamma_{nk} \triangleq p(\mathcal{C}_k|x_n) = p(z_{nk} = 1|x_n)$

ML estimation for Clustering: The Expectation-Maximization (EM) Algorithm Idea

- **IDEA**: Let's apply the (generative) classification formulas and substitute the reponsibilities γ_{nk} wherever the formulas use the binary class indicators y_{nk}.
- Try parameter updates (like generative Gaussian classification):

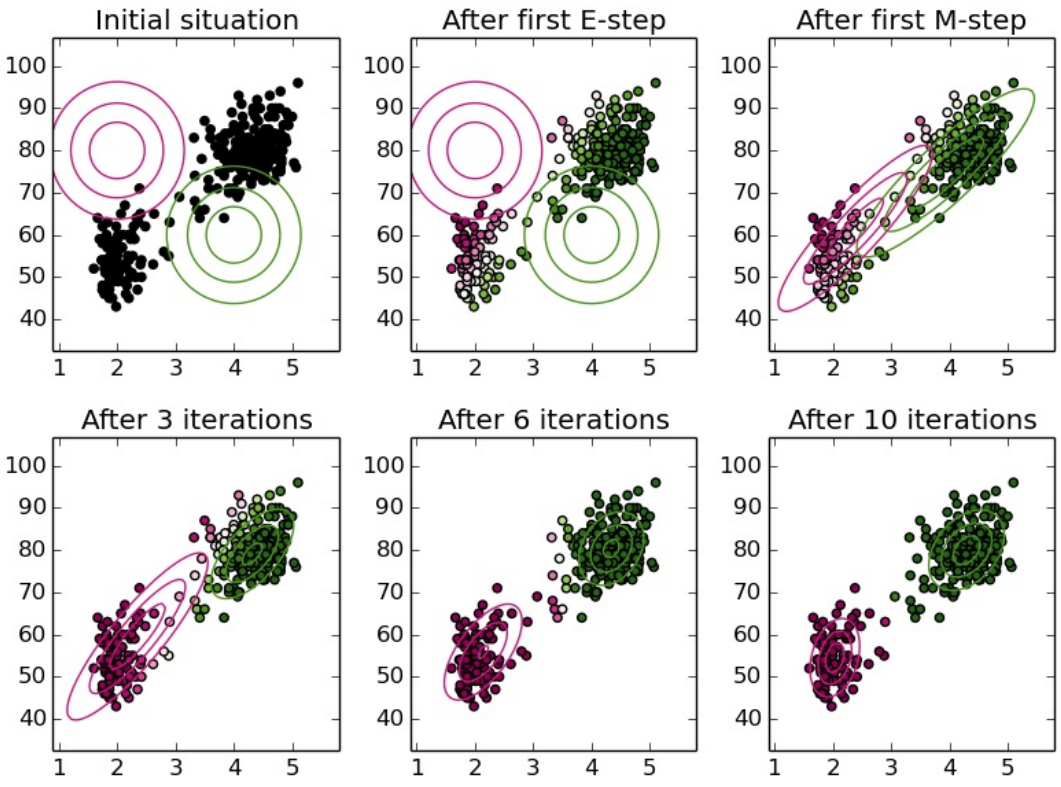
$$\hat{\pi}_k = \frac{m_k}{N}$$
$$\hat{\mu}_k = \frac{1}{m_k} \sum_n \gamma_{nk} x_n$$
$$\hat{\Sigma}_k = \frac{1}{m_k} \sum_n \gamma_{nk} (x_n - \hat{\mu}_k)(x_n - \hat{\mu}_k)^T$$

  where $m_k = \sum_n \gamma_{nk}$.
- But wait ..., the responsibilities $\gamma_{nk} = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_j \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)}$ are a function of the model parameters $\{\pi, \mu, \Sigma\}$ and the parameter updates depend on the responsibilities ...
- **Solution(?)**: iterate between updating the responsibilities γ_{nk} and the model parameters $\{\pi, \mu, \Sigma\}$.
- This iteration works (!) and is called the **Expectation-Maximization (EM)** algorithm.

CODE EXAMPLE

We'll perform clustering on the data set from the illustrative example by fitting a GMM consisting of two Gaussians using the EM algorithm.

```
using DataFrames
include("scripts/gmm_plot.jl") # Holds plotting function
old_faithful = readtable("../files/datasets/old_faithful.csv")
X = convert(Matrix{Float64}, [old_faithful[1] old_faithful[2]]')
N = size(X, 2)

# Initialize the GMM. We assume 2 clusters.
clusters = [MvNormal([4.;60.], diagm([.5;10^2]));
            MvNormal([2.;80.], diagm([.5;10^2]))]
π_hat = [0.5; 0.5]                    # Mixing weights
γ = fill!(Matrix{Float64}(2,N), NaN)  # Responsibilities (row per cluster)

# Define functions for updating the parameters and responsibilities
function updateResponsibilities!(X, clusters, π_hat, γ)
    # Expectation step: update γ
    norm = [pdf(clusters[1], X) pdf(clusters[2], X)] * π_hat
    γ[1,:] = (π_hat[1] * pdf(clusters[1],X) ./ norm)'
    γ[2,:] = 1 - γ[1,:]
end
function updateParameters!(X, clusters, π_hat, γ)
    # Maximization step: update π_hat and clusters using ML estimation
    m = sum(γ, 2)
    π_hat = m / N
    μ_hat = (X * γ') ./ m'
    for k=1:2
        Σ_k = (((X .- μ_hat[:,k]) .* γ[k,:]) * (X .- μ_hat[:,k])') / m[k]
        clusters[k] = MvNormal(μ_hat[:,k], Σ_k)
    end
end

# Execute the algorithm: iteratively update parameters and responsibilities
subplot(2,3,1); plotGMM(X, clusters, γ); title("Initial situation")
updateResponsibilities!(X, clusters, π_hat, γ)
subplot(2,3,2); plotGMM(X, clusters, γ); title("After first E-step")
updateParameters!(X, clusters, π_hat, γ)
subplot(2,3,3); plotGMM(X, clusters, γ); title("After first M-step")
iter_counter = 1
for i=1:3
    for j=1:i+1
        updateResponsibilities!(X, clusters, π_hat, γ)
        updateParameters!(X, clusters, π_hat, γ)
        iter_counter += 1
    end
    subplot(2,3,3+i); plotGMM(X, clusters, γ);
    title("After $(iter_counter) iterations")
end
PyPlot.tight_layout()
```



Note that you can step through the interactive demo yourself by running this script in julia. You can run a script in julia by
`julia> include("path/to/script-name.jl")`

END OF CODE EXAMPLE

Clustering vs. (Generative) Classification

	Classification	Clustering
1	Class label y_n is observed	Class label z_n is latent
2	log-likelihood **conditions** on observed class $$\propto \sum_{nk} y_{nk} \log \mathcal{N}(x_n \mid \mu_k, \Sigma_k)$$	log-likelihood **marginalizes** over latent classes $$\propto \sum_{n} \log \sum_{k} \pi_k \mathcal{N}(x_n \mid \mu_k, \Sigma_k)$$
3	'Hard' class selector $$y_{nk} = \begin{cases} 1 & \text{if } y_n \text{ in class } \mathcal{C}_k \\ 0 & \text{otherwise} \end{cases}$$	'Soft' class responsibility $$\gamma_{nk} = p(\mathcal{C}_k \mid x_n)$$
4	Estimation: $$\hat{\pi}_k = \frac{1}{N} \sum_n y_{nk}$$ $$\hat{\mu}_k = \frac{\sum_n y_{nk} x_n}{\sum_n y_{nk}}$$ $$\hat{\Sigma}_k = \frac{\sum_n y_{nk} (x_n - \hat{\mu}_k)(x_n - \hat{\mu}_k)^T}{\sum_n y_{nk}}$$	Estimation (1 update of M-step!) $$\hat{\pi}_k = \frac{1}{N} \sum_n \gamma_{nk}$$ $$\hat{\mu}_k = \frac{\sum_n \gamma_{nk} x_n}{\sum_n \gamma_{nk}}$$ $$\hat{\Sigma}_k = \frac{\sum_n \gamma_{nk} (x_n - \hat{\mu}_k)(x_n - \hat{\mu}_k)^T}{\sum_n \gamma_{nk}}$$

The General EM Algorithm

Preliminaries

- Goal
  - A more formal treatment of the general EM algorithm
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 55-57 for Jensen's inequality
    - Bishop pp. 439-443 for EM applied to GMM
    - Bishop pp. 450-455 for the general EM algorithm
    - Liang (2015) Technical Details about the EM algorithm

The Kullback–Leibler Divergence

- In order to prove that the EM algorithm works, we will need Gibbs inequality, which is a famous theorem in information theory.
- Definition: the **Kullback-Leibler divergence** (a.k.a. **relative entropy**) is a distance measure between two distributions q and p and is defined as

$$D_{\mathrm{KL}}(q \parallel p) \triangleq \sum_z q(z) \log \frac{q(z)}{p(z)}$$

- Theorem: **Gibbs Inequality** (proof uses Jensen inquality):

$$D_{\mathrm{KL}}(q \parallel p) \geq 0$$

  with equality only iff $p = q$.
- Note that the KL divergence is an asymmetric distance measure, i.e. in general

$$D_{\mathrm{KL}}(q \parallel p) \neq D_{\mathrm{KL}}(p \parallel q)$$

EM as Free Energy minimization

- Consider a model for observations x, hidden variables z and tuning parameters θ. Note that, for **any** distribution $q(z)$, we can expand the log-likelihood as follows:

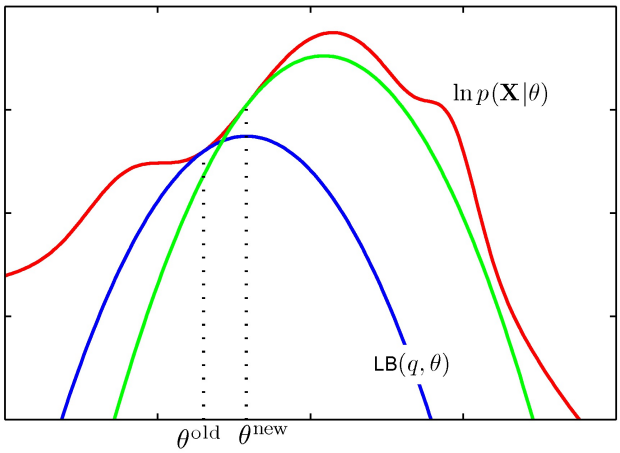
$$\begin{aligned}
\mathrm{L}(\theta) &\triangleq \log p(x|\theta) \\
&= \sum_z q(z) \log p(x|\theta) \\
&= \sum_z q(z) \left(\log p(x|\theta) - \log \frac{q(z)}{p(z|x,\theta)} \right) + \sum_z q(z) \log \frac{q(z)}{p(z|x,\theta)} \\
&= \sum_z q(z) \log \frac{p(x,z|\theta)}{q(z)} + \underbrace{D_{\mathrm{KL}}\left(q(z) \parallel p(z|x,\theta)\right)}_{\text{Kullback-Leibler div.}} \\
&\geq \sum_z q(z) \log \frac{p(x,z|\theta)}{q(z)} \quad \text{(use Gibbs inequality)} \\
&= \underbrace{\sum_z q(z) \log p(x,z|\theta)}_{\text{expected complete-data log-likelihood}} + \underbrace{\mathcal{H}[q]}_{\text{entropy of } q} \\
&\triangleq \mathrm{LB}(q,\theta)
\end{aligned}$$

- $\mathrm{LB}(q,\theta)$ is a **lower bound** on the log-likelihood $\log p(x|\theta)$.
- Technically, the Expectation-Maximization (EM) algorithm is defined by coordinate ascent on the lower-bound $\mathrm{LB}(q,\theta)$:

$$\begin{aligned}
&\text{Initialize } : \theta^{(0)} \\
&\text{for } m = 1,2,\ldots \text{until convergence} \\
&\quad q^{(m+1)} = \arg\max_q \mathrm{LB}(q,\theta^{(m)}) \\
&\quad \theta^{(m+1)} = \arg\max_\theta \mathrm{LB}(q^{(m+1)},\theta)
\end{aligned}$$

  where m is the iteration counter.

- Since $\mathrm{LB}(q,\theta) \leq \mathrm{L}(\theta)$ (always), maximizing the lower-bound LB will also maximize the log-likelihood. The *reason* to maximize LB rather than log-likelihood L directly is that $\arg\max \mathrm{LB}$ often leads to easier expressions. E.g., see this illustrative figure:



- Just as an aside, the negative lower-bound $\mathrm{F}(q,\theta) \triangleq -\mathrm{LB}(q,\theta)$ also appears in various scientific disciplines. In statistical physics and variational calculus, F is known as the **free energy** functional. Hence, the EM algorithm is a special case of **free energy minimization**.
  - A very influential neuroscientific theory claims that information processing in the brain is also an example of free-energy minimization, see Friston, 2009.

EM Details

Maximizing $\mathbf{LB}$ w.r.t. q

- Note that
$$\mathrm{LB}(q,\theta) = \mathrm{L}(\theta) - D_{\mathrm{KL}}\left(q(z) \parallel p(z|x,\theta)\right)$$
and consequenty, maximizing LB over q leads to minimization of the KL-divergence. Specifically, it follows from Gibbs inequality that
$$q^{(m+1)}(z) := p(z|x,\theta^{(m)})$$

Maximizing $\mathbf{LB}$ w.r.t. θ

- It also follows from (the last line of) the multi-line derivation above that maximizing LB w.r.t. θ amounts to maximization of the *expected complete-data log-likelihood*. Hence, the EM algorithm comprises iterations of
$$\mathbf{EM}: \ \theta^{(m+1)} := \underbrace{\arg\max_{\theta}}_{\text{M-step}} \underbrace{\sum_{z} \overbrace{p(z|x,\theta^{(m)})}^{=q^{(m+1)}(z)} \log p(x,z|\theta)}_{\text{E-step}}$$

Exercise: EM for GMM revisited

E-step

- Write down the GMM generative model
- The complete-data set is $D_c = \{x_1, z_1, x_2, z_2, \ldots, x_n, z_n\}$. Write down the *complete-data* likelihood $p(D_c|\theta)$
- Write down the complete-data *log*-likelihood $\log p(D_c|\theta)$
- Write down the *expected* complete-data log-likelihood $\mathrm{E}_Z\left[\log p(D_c|\theta)\right]$

M-step

- Maximize $\mathrm{E}_Z\left[\log p(D_c|\theta)\right]$ w.r.t. $\theta = \{\pi, \mu, \Sigma\}$
- Verify that your solution is the same as the 'intuitive' solution of the previous lesson.

Exercise: EM for Three Coins problem

- You have three coins in your pocket. For each coin, outcomes $\in \{\mathrm{H}, \mathrm{T}\}$.
$$p(\mathrm{H}) = \begin{cases} \lambda & \text{for coin } 0 \\ \rho & \text{for coin } 1 \\ \theta & \text{for coin } 2 \end{cases}$$
- **Scenario**. Toss coin 0. If Head comes up, toss three times with coin 1; otherwise, toss three times with coin 2.
- The observed sequences **after** each toss with coin 0 were $\langle \mathrm{HHH} \rangle$, $\langle \mathrm{HTH} \rangle$, $\langle \mathrm{HHT} \rangle$, and $\langle \mathrm{HTT} \rangle$
- **Task**. Use EM to estimate most the likely values for λ, ρ and θ

Some Properties of EM

- EM is a general procedure for learning in the presence of unobserved variables.
- In a sense, it is a **family of algorithms**. The update rules you will derive depend on the probability model assumed.
- (Good!) **No tuning parameters** such a learning rate, unlike gradient descent-type algorithms
- (Bad). EM is an iterative procedure that is very sensitive to initial conditions! EM converges to a **local optimum**.
- Start from trash $\longrightarrow$ end up with trash. Hence, we need a good and fast initialization procedure (often used: K-Means)
- Also used to train HMMs, etc.

Continuous Latent Variable Models – PCA and FA

Preliminaries

- Goal
  - Introduction to Linear Latent Variable Models on continuous domains, specifically **factor analysis** and **principal component analysis**
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 570-573, 577-580, 584-586 (PCA and FA)
    - M. Tipping and C. Bishop, Probabilistic Principal Component Analysis, Journal of the Royal Statistical Society. Series B, Vol.61, No.3, 1999

Continuous Latent Variable Models

- (Recall that) mixture models use a discrete class variable.
- Sometimes, it is more appropriate to think in terms of **continuous** underlying causes (factors) that control the observed data.
  - E.g., observe test results for subjects: English, Spanish and French

$$\underbrace{\begin{bmatrix} x_1\ (=\text{English}) \\ x_2\ (=\text{Spanish}) \\ x_3\ (=\text{French}) \end{bmatrix}}_{\text{observed}} = \begin{bmatrix} \lambda_{11}, \lambda_{12} \\ \lambda_{21}, \lambda_{22} \\ \lambda_{31}, \lambda_{32} \end{bmatrix} \cdot \underbrace{\begin{bmatrix} z_1\ (=\text{literacy}) \\ z_2\ (=\text{intelligence}) \end{bmatrix}}_{\text{causes}} + \underbrace{\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}}_{\text{noise}}$$

- (**Unsupervised Regression**). This is like (linear) regression with unobserved inputs.

Dimensionality Reduction

- If the dimension for the hidden 'causes' (z) is smaller than for the observed data (x), then the model (tries to) achieve **dimensionality reduction**.
- Key applications include
  1. **compression** (store z rather than x)
     - Compression through **real-valued** latent variables can be far more efficient than with discrete clusters.
     - E.g., with two 8-bit hidden factors, one can describe $2^{16} \approx 10^5$ settings; this would take 2^{16} clusters!
  2. **noise reduction** (e.g. in biomedical, financial or speech signals)
  3. **feature extraction** (e.g. as a pre-processor for classification)
  4. **visualization** (particularly if $\dim(Z) = 2$)

Example Problem: Visualization with missing data

- We consider 38 examples from the 18-dimensional data set from the *Tobamovirus* data set, see section 4.1 in Tipping and Bishop (1999) (Originally from Ripley (1996)).
- We will visualize this data set after projection onto the two principal axes (i.e., axes that explain largest data variance).
- We will also consider the visualization problem when **20% of the data set is missing**.

```
include("scripts/pca_demo_helpers.jl")
X = readDataSet("../files/datasets/virus3.dat") # Read the data set
dump(X); # Show the raw 18-dimensional data
```

```
Array(Int64,(38,18)) 38x18 Array{Int64,2}
```

Model Specification for LC-LVM

- In this lesson, we focus on *Linear* Continuous Latent Variable Models (**LC-LVM**).
- Introduce observation vector $x \in \mathbb{R}^D$ and M-dimensional (with $M < D$) real-valued **latent factor** z:

$$\begin{aligned} x &= Wz + \mu + \epsilon \\ z &\sim \mathcal{N}(0, I) \\ \epsilon &\sim \mathcal{N}(0, \Psi) \end{aligned}$$

  or equivalently

$$\begin{aligned} p(x|z,\theta) &= \mathcal{N}(x|\, Wz + \mu, \Psi) && \text{(likelihood)} \\ p(z) &= \mathcal{N}(z|\, 0, I) && \text{(prior)} \end{aligned}$$

  where W is the $(D \times M)$-dim **factor loading matrix**
- As we will see, the **observation noise covariance matrix** Ψ is always **diagonal** for interesting LC-LVM models.
- Note that the likelihood function in LC-LVM is very similar to (the likelihood fucntion in) linear regression:

$$p(y|x) = \mathcal{N}(y|\theta^T x, \sigma^2)$$

- Note also that the components of the hidden variables Z are **statistically independent** of each other; the components of the observed vector X may be correlated.

LC-LVM Analysis (1): The marginal distribution $p(x)$

- Since the product of Gaussians is Gaussian, both the joint $p(x,z) = p(x|z)p(z)$, the marginal $p(x)$ and the conditional $p(z|x)$ distributions are also Gaussian.
- The marginal distribution for the observed data is

$$p(x) = \mathcal{N}(x|\,\mu, WW^T + \Psi)$$

since the **mean** evaluates to

$$\begin{aligned}\mathrm{E}[x] &= \mathrm{E}[Wz + \mu + \epsilon] \\ &= W\mathrm{E}[z] + \mu + \mathrm{E}[\epsilon] \\ &= \mu\end{aligned}$$

and the **covariance** matrix is

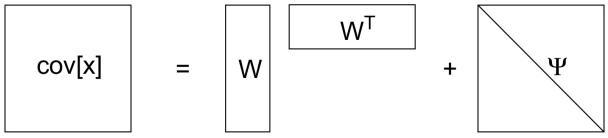
$$\begin{aligned}\mathrm{cov}[x] &= \mathrm{E}[(x-\mu)(x-\mu)^T] \\ &= \mathrm{E}[(Wz+\epsilon)(Wz+\epsilon^T)] \\ &= W\mathrm{E}[zz^T]W^T + \mathrm{E}[\epsilon\epsilon^T] \\ &= WW^T + \Psi\end{aligned}$$

⇒ **LC-LVM is just a MultiVariate Gaussian (MVG) model** $x \sim \mathcal{N}(\mu, \Sigma)$ with the restriction that

$$\Sigma = WW^T + \Psi$$

.

- The effective covariance is the low-rank outer product of two long skinny matrices plus a diagonal matrix.



⇒ LC-LVM provides a MVG model of **intermediate complexity**. Compare the number of free parameters:

- $D(D+1)/2$ for full Gaussian covariance Σ
- $D(M+1)$ for LC-LVM model where $\Sigma = WW^T + \Psi$.
- D for diagonal Gaussian covariance $\Sigma = \mathrm{diag}(\sigma_i^2)$
- 1 for isotropic Gaussian noise $\Sigma = \sigma^2\mathrm{I}$

LC-LVM Analysis (2): The Factor Loading Matrix W is Not Unique

- The factor loading matrix W can only be estimated up to a rotation matrix R. Namely, if we rotate $W \to WR$, then the covariance matrix for observations x does not change (N.B.: a rotation (or orthogonal) matrix R is a matrix such that $R^TR = RR^T = I$):

$$WR(WR)^T + \Psi = WRR^TW^T + \Psi = WW^T + \Psi$$

⇒ Two persons that estimate ML parameters for FA on the same data are **not guaranteed to find the same parameters**, since any rotation of W is equally likely.

⇒ we can infer latent **subspaces** rather than individual components. One has to be careful when interpreting the numerical values of W and z.

LC-LVM analysis (3): Constraints on the Noise Variance $\mathbf{\Psi}$

- When doing ML estimation for the parameters, a trivial solution for the covariance matrix $\Sigma_x = WW^T + \Psi$ is setting $\hat{W} = 0$ and $\hat{\Psi}$ equal to the sample variance of the data.
- In this case, all data correlation is explained as noise. (We'd like to avoid this).

$\Rightarrow$ The LC-LVM model is uninteresting without some restriction on the observation noise covariance matrix Ψ.

- The interesting cases are mostly for diagonal Ψ. Note that if Ψ is diagonal, all correlations between the (D) components of x **must be explained** by the rank-M matrix WW^T.

Factor Anaysis

- In Factor Analysis (**FA**), Ψ is restricted to be *diagonal*:

$$\Psi = \mathrm{diag}(\psi_i)$$

Probabilistic Principal Component Analysis

- In Probabilistic Principal Component Analysis (**pPCA**), the variances are further restricted to be the same,

$$\Psi = \sigma^2 I$$

Principal Component Analysis

- The 'regular' (deterministic) Principal Component Analysis (**PCA**) procedure can be obtained by further requiring that

$$\Psi = \lim_{\sigma^2 \to 0} \sigma^2 I$$
$$W^T W = I$$

  i.e., the noise model is discarded altogether and the columns of W are orthonormal.
- Regular PCA is a well-known deterministic procedure for dimensionality reduction (that predates pPCA).

$\Rightarrow$ FA, pPCA and PCA differ by their model for the noise variance Ψ (namely, diagonal, isotropic and 'zeros').

Typical Applications

- In PCA (or pPCA), the noise variance is assumed to be the same for all components. This is appropriate if all components of the observed data are 'shifted' versions of each other.

$\Rightarrow$ **PCA is very widely applied to image and signal processing tasks!**

- Google (May-2015): [PCA "face recognition"] $>$ 300K hits; [PCA "noise reduction"] $>$ 100K hits
- FA is insensitive to scaling of individual components in the observed data (see appendix).
- Use FA if the data are not shifted versions of the same kind.

$\Rightarrow$ **FA has strong history in 'social sciences'**

ML estimation for pPCA Model

- Given the generative model for pPCA

$$p(x_n|z_n) = \mathcal{N}(x_n \mid W z_n + \mu, \sigma^2 \mathrm{I})$$
$$p(z_n) = \mathcal{N}(z_n \mid 0, \mathrm{I})$$

  and observations $D = \{x_1, \ldots, x_N\}$, find ML estimates for the parameters $\theta = \{W, \mu, \sigma\}$
- **Inference for** μ is easy: x is a multivariate Gaussian with mean μ, so its ML estimate is

$$\hat{\mu} = \frac{1}{N} \sum_n x_n$$

  Now subtract $\hat{\mu}$ from all data points ($x_n := x_n - \hat{\mu}$) and assume that we have zero-mean data.
- For ML estimation of W and σ^2, both gradient-ascent and EM are possible.

Solution method 1: Gradient-ascent on the log-likelihood

- Work out the gradients for the log-likelihood

$$\begin{aligned}&\log p(D|\theta)\\&\quad=-\frac{N}{2}\log|2\pi(WW^T+\sigma^2\mathrm{I})|-\frac{1}{2}\sum_n x_n^T(WW^T+\sigma^2\mathrm{I})^{-1}x_n\end{aligned}$$

  and optimize w.r.t. W and σ^2.

- (Similarly to ML estimation in Gaussian mixture models), it turns out to be quite difficult to work out the gradient because of the coupling between W and σ^2 (but it is possible, see Tipping and Bishop, 1999).

Solution method 2: Use EM

- A big bonus for EM over gradient-based methods is that EM comfortably handles missing observations, e.g. through sensor malfunction. Missing observations are simply treated as hidden variables.

- Maximizing the *expected complete-data log-likelihood* leads to the following (see Bishop, pg.578 for derivation):

$$\begin{aligned}&\textbf{E-step}:\\&M=W^TW+\sigma^2\mathrm{I}\\&\mathrm{E}[z_n]=M^{-1}W^Tx_n\\&\mathrm{E}\left[z_nz_n^T\right]=\sigma^2M^{-1}+\mathrm{E}[z_n]\mathrm{E}[z_n]^T\end{aligned}$$

$$\begin{aligned}&\textbf{M-step}:\\&W_{\text{new}}=\left[\sum_{n=1}^N x_n\mathrm{E}[z_n]^T\right]\left[\sum_{n=1}^N\mathrm{E}\left[z_nz_n^T\right]\right]^{-1}\\&\sigma^2_{\text{new}}=\frac{1}{ND}\sum_{n=1}^N\left\{x_n^Tx_n-2\mathrm{E}[z_n]^TW_{\text{new}}^Tx_n+\mathrm{Tr}\left(\mathrm{E}\left[z_nz_n^T\right]W_{\text{new}}^TW_{\text{new}}\right)\right\}\end{aligned}$$

- Note that after x_n is observed, the unobserved 'input' z_n is not known exactly; the uncertainty about input z_n, as expressed by the covariance

$$\mathrm{cov}(z_n)=\mathrm{E}\left[z_nz_n^T\right]-\mathrm{E}[z_n]\mathrm{E}[z_n]^T=\sigma^2M^{-1}$$

  can be computed _before the data point x_n has been seen_. Exercise: Show that the precision about z_n increases through observing x_n.

  - Compare this to linear regression, where we have full knowledge about an input-output pair.

- If there was no uncertainty about z_n, i.e., $\mathrm{E}[z_n]=z_n$ and $\mathrm{E}\left[z_nz_n^T\right]=z_nz_n^T$, then

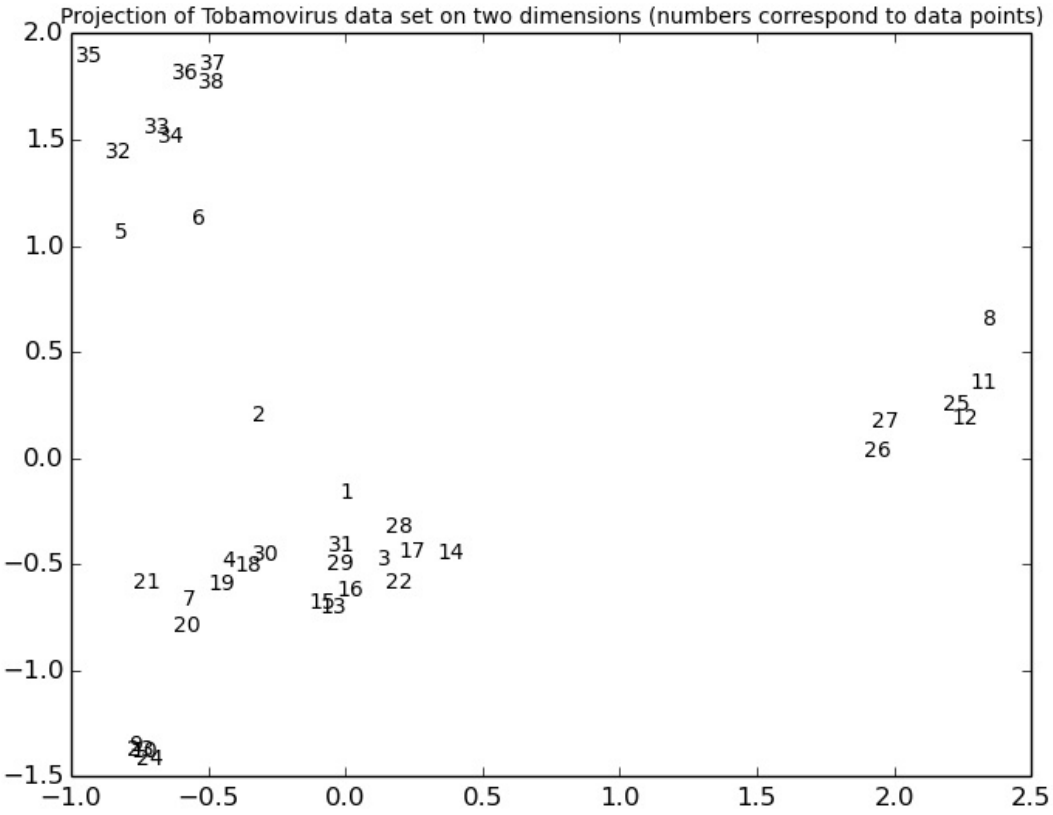
$$W_{\text{new}}=\left[\sum_{n=1}^N x_nz_n^T\right]\left[\sum_{n=1}^N z_nz_n^T\right]^{-1}$$

  - Verify that this solution resembles the ML solution for linear regression.

Example Problem Revisited

Let's perform pPCA on the example (*Tobamovirus*) data set using EM. We'll find the two principal components ($M=2$), and then visualize the data in a 2-D plot. The implementation is quite straightforward, have a look at the source file if you're interested in the details.

```
(θ, Z) = pPCA(X', 2) # uses EM, implemented in scripts/pca_demo_helpers.jl. Feel free to try more/less dimensions.
using PyPlot
plot(Z[1,:]', Z[2,:]', "w")
for n=1:size(Z,2)
    PyPlot.text(Z[1,n], Z[2,n], string(n), fontsize=10) # put a label on the position of the data point
end
title("Projection of Tobamovirus data set on two dimensions (numbers correspond to data points)", fontsize=10);
```



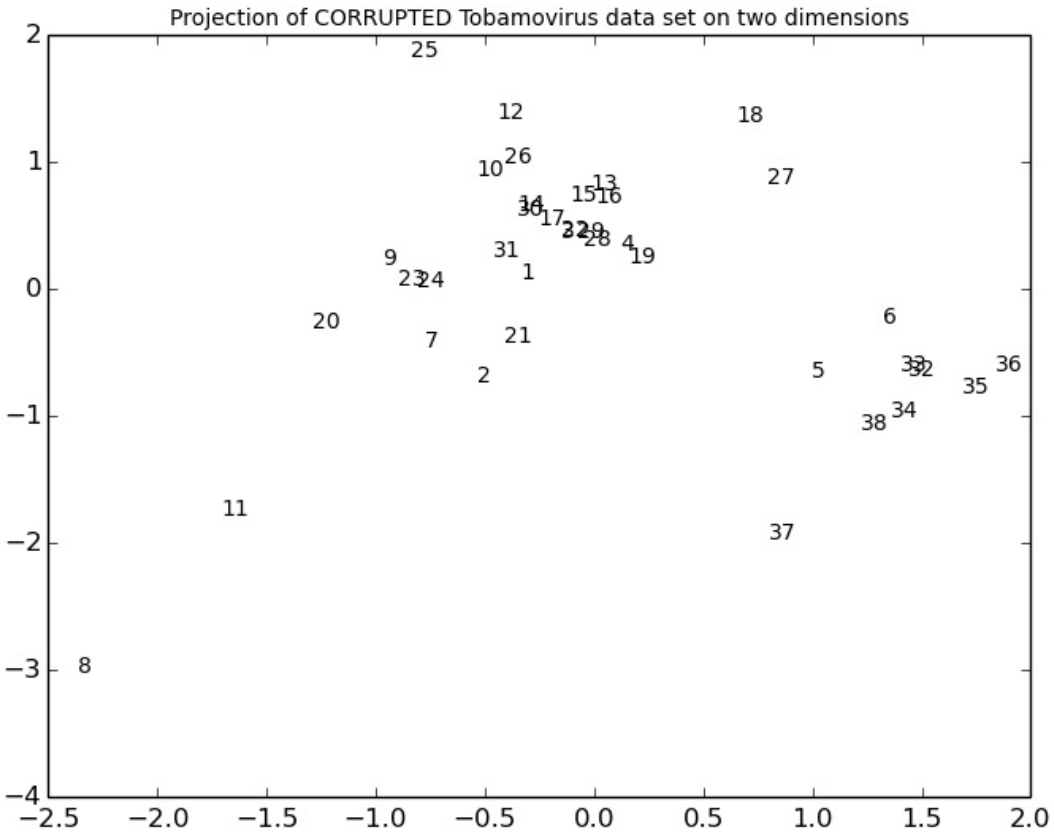
Note that the solution is not unique, but the clusters should be more or less persistent.

Now let's randomly remove 20% of the data:

```
X_corrupt = convert(Matrix{Float64}, X) # convert to floating point matrix so we can use NaN to indicate missing values
X_corrupt[rand(length(X)) .< 0.2] = NaN # set the elements of X_corrupt to NaN with probability 0.2
dump(X_corrupt)
```

```
Array(Float64,(38,18)) 38x18 Array{Float64,2}
```

```
(θ, Z) = pPCA(X_corrupt', 2) # Perform pPCA on the corrupted data set
plot(Z[1,:]', Z[2,:]', "w")
for n=1:size(Z,2)
    PyPlot.text(Z[1,n], Z[2,n], string(n), fontsize=10) # put a label on the position of the data
point
end
title("Projection of CORRUPTED Tobamovirus data set on two dimensions", fontsize=10);
```



As you can see, pPCA is quite robust in the face of missing data.

The cell below loads the style file

Factor Graphs

Preliminaries

- Goal
  - Introduction to Forney-style factor graphs and message passing algorithms
- Materials
  - Mandatory
    - These lecture notes
    - Loeliger, 2007, pp. 1295-1300
  - Optional
    - Video lecture by Frederico Wadehn (ETH Zurich)

Why Factor Graphs?

- A probabilistic inference task gets its computational load mainly through the need for marginalization (i.e., computing integrals). E.g., for a generative model $p(x_1, x_2, x_3, x_4, x_5)$, the inference task $p(x_2|x_3)$ is given by

$$p(x_2|x_3) = \frac{\int p(x_1, x_2, x_3, x_4, x_5)\, \mathrm{d}x_1 \mathrm{d}x_4 \mathrm{d}x_5}{\int p(x_1, x_2, x_3, x_4, x_5)\, \mathrm{d}x_1 \mathrm{d}x_2 \mathrm{d}x_4 \mathrm{d}x_5}$$

- Since these computations suffer from the "curse of dimensionality", we often need to solve a simpler problem in order to get an answer.
- Factor graphs provide an computationally efficient approach to solving inference problems **if the generative distribution can be factorized**.
- Factorization helps. For instance, if $p(x_1, x_2, x_3, x_4, x_5) = p(x_1)p(x_2, x_3)p(x_4)p(x_5|x_4)$, then

$$\begin{aligned} p(x_2|x_3) &= \frac{\int p(x_1)p(x_2, x_3)p(x_4)p(x_5|x_4)\, \mathrm{d}x_1 \mathrm{d}x_4 \mathrm{d}x_5}{\int p(x_1)p(x_2, x_3)p(x_4)p(x_5|x_4)\, \mathrm{d}x_1 \mathrm{d}x_2 \mathrm{d}x_4 \mathrm{d}x_5} \\ &= \frac{p(x_2, x_3)}{\int p(x_2, x_3) \mathrm{d}x_2} \end{aligned}$$

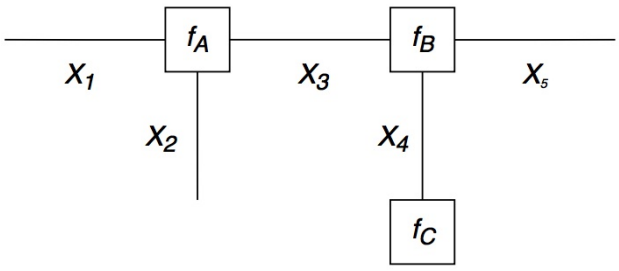
  which is computationally much cheaper than the general case above.
- In this lesson, we discuss how computationally efficient inference in factorized probability distributions can be automated.

Construction Rules

- Consider a function

$$f(x_1, x_2, x_3, x_4, x_5) = f_a(x_1, x_2, x_3) \cdot f_b(x_3, x_4, x_5) \cdot f_c(x_4)$$

- The factorization of this function can be graphically represented by a **Forney-style Factor Graph** (FFG):



- An FFG is an **undirected** graph subject to the following construction rules (Forney, 2001)

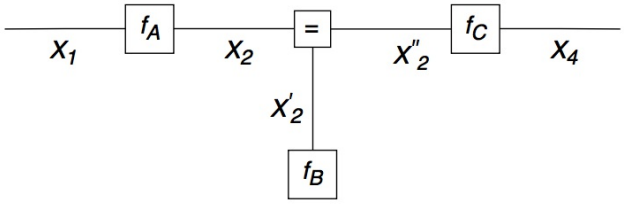
1. A **node** for every factor;
2. An **edge** (or **half-edge**) for every variable;
3. Node g is connected to edge x **iff** variable x appears in factor g.

some terminology

- f is called the **global function** and $f_\bullet$ are the **factors**.
- A **configuration** is an assigment of values to all variables.
- The **configuation space** is the set of all configutations, i.e., the domain of f
- A configuration $\omega = (x_1, x_2, x_3, x_4, x_5)$ is said to be **valid** iff $f(\omega) \neq 0$

Equality Nodes for Branching Points

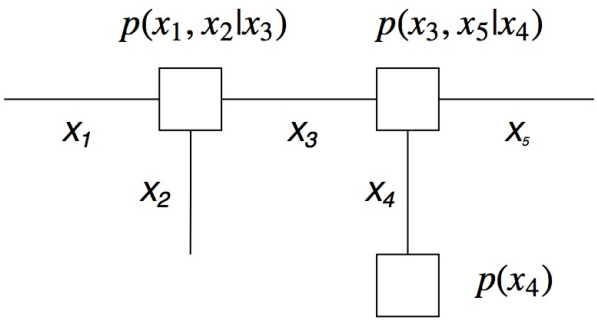
- Note that a variable can appear in maximally two factors in an FFG.
- Consider the factorization (where x_2 appears in three factors)
$$f(x_1, x_2, x_3, x_4) = f_a(x_1, x_2) \cdot f_b(x_2, x_3) \cdot f_c(x_2, x_4)$$
- For the factor graph representation, we will instead consider the function g, defined as
$$\begin{aligned} &g(x_1, x_2, x_2', x_2'', x_3, x_4) \\ &\quad = f_a(x_1, x_2) \cdot f_b(x_2', x_3) \cdot f_c(x_2'', x_4) \cdot f_=(x_2, x_2', x_2'') \end{aligned}$$
where
$$f_=(x_2, x_2', x_2'') \triangleq \delta(x_2 - x_2')\,\delta(x_2 - x_2'')$$
(where δ is the Kronecker delta for discrete variables and the Dirac delta for continuously valued variables)
- Note that through introduction of auxiliary variables X_2' and X_2'' each variable in g appears in maximally two factors.
- The constraint $f_=(x, x', x'')$ enforces that $X = X' = X''$ **for every valid configuration**.



- Also note that f is a marginal of g, since
$$f(x_1, x_2, x_3, x_4) = \int g(x_1, x_2, x_2', x_2'', x_3, x_4)\, \mathrm{d}x_2'\, \mathrm{d}x_2''$$
- Therefore, any inference problem on f can be executed by a corresponding inference problem on g, e.g.,
$$\begin{aligned} f(x_1 \mid x_2) &\triangleq \frac{\int f(x_1, x_2, x_3, x_4)\, \mathrm{d}x_3\, \mathrm{d}x_4}{\int f(x_1, x_2, x_3, x_4)\, \mathrm{d}x_1 \mathrm{d}x_3 \mathrm{d}x_4} \\ &= \frac{\int g(x_1, x_2, x_2', x_2'', x_3, x_4)\, \mathrm{d}x_2' \mathrm{d}x_2'' \mathrm{d}x_3 \mathrm{d}x_4}{\int g(x_1, x_2, x_2', x_2'', x_3, x_4)\, \mathrm{d}x_1 \mathrm{d}x_2' \mathrm{d}x_2'' \mathrm{d}x_3 \mathrm{d}x_4} \\ &\triangleq g(x_1 \mid x_2) \end{aligned}$$
- $\Rightarrow$ **Any factorization of a global function f can be represented by a Forney-style Factor Graph**.
- More generally, equality nodes are useful to express hard constraints between variables.

Probabilistic Models as Factor Graphs

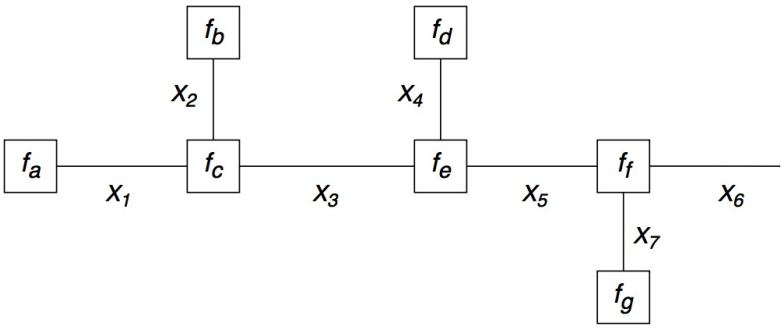
- FFGs can be used to express conditional independence (factorization) in probalistic models.
- For example, the (previously shown) graph for
$$f_a(x_1,x_2,x_3)\cdot f_b(x_3,x_4,x_5)\cdot f_c(x_4)$$
could represent the probabilistic model
$$p(x_1,x_2,x_3,x_4,x_5) = p(x_1,x_2|x_3)\cdot p(x_3,x_5|x_4)\cdot p(x_4)$$
where we identify
$$\begin{aligned} f_a(x_1,x_2,x_3) &= p(x_1,x_2|x_3)\\ f_b(x_3,x_4,x_5) &= p(x_3,x_5|x_4)\\ f_c(x_4) &= p(x_4)\end{aligned}$$
- This is the graph



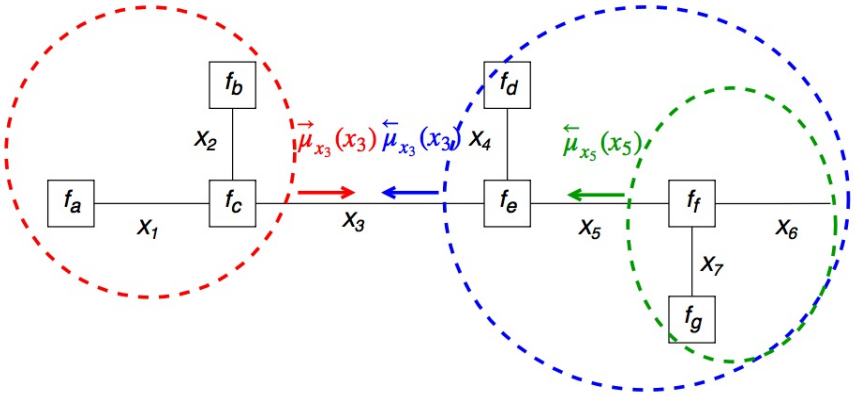
- Factorizations provide opportunities to cut on the amount of needed computations when doing inference. In what follows, we will use FFGs to process these opportunities in an automatic way (i.e., by message passing).

Inference by Closing Boxes

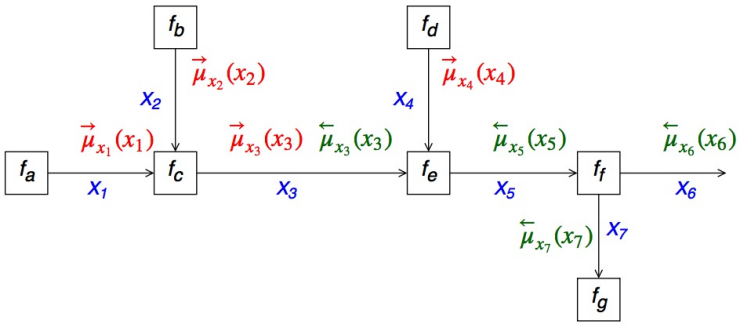
- Assume we wish to compute the marginal
$$\bar{f}(x_3) = \sum_{x_1,x_2,x_4,x_5,x_6,x_7} f(x_1,x_2,\ldots,x_7)$$
where f is factorized as given by the following FFG



- Due to the factorization, we can decompose this sum by the distributive law as
$$\bar{f}(x_3) = \underbrace{\left(\sum_{x_1,x_2} f_a(x_1)\, f_b(x_2)\, f_c(x_1,x_2,x_3)\right)}_{\vec{\mu}_{X_3}(x_3)} \cdot \underbrace{\left(\sum_{x_4,x_5} f_d(x_4)\, f_e(x_3,x_4,x_5)\cdot \underbrace{\left(\sum_{x_6,x_7} f_f(x_5,x_6,x_7)\, f_g(x_7)\right)}_{\overleftarrow{\mu}_{X_5}(x_5)}\right)}_{\overleftarrow{\mu}_{X_3}(x_3)}$$
which is computationally (much) lighter than executing the full sum $\sum_{x_1,\ldots,x_7} f(x_1,x_2,\ldots,x_7)$



- Note that messages may flow in both directions on any particular edge (here on X_3). We often draw *directed edges* in a FFG in order to distinguish forward messages $\vec{\mu}_\bullet(\cdot)$ (in the same direction as the arrow of the edge) from backward messages $\overleftarrow{\mu}_\bullet(\cdot)$ (in opposite direction). With directed edges, the FFG looks as follows:



- Crucially, drawing arrows on edges is only meant as a notational convenience. Technically, an FFG is an undirected graph.
- Also note that the message $\overleftarrow{\mu}_{X_5}(x_5)$ is obtained by _multiplying all factors that are enclosed by the dashed boxes (f_f and f_g), followed by marginalization over all enclosed variables_ (x_6 and x_7).
- This is the "Closing the Box"-rule, which is a general recipe for marginalization of hidden variables and leads to a new factor with outgoing (sum-product) message

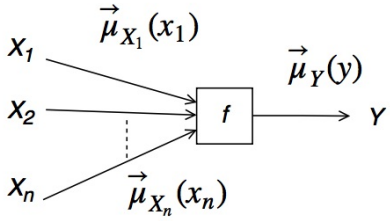
$$\mu_{\text{SP}} = \sum_{\substack{\text{enclosed} \\ \text{variables}}} \prod_{\substack{\text{enclosed} \\ \text{factors}}}$$

- Let's evaluate the closing-the-box rule for individual nodes.
  - First, closing a box around the 'terminal' nodes leads to $\vec{\mu}_{X_1}(x_1) \triangleq f_a(x_1)$, $\vec{\mu}_{X_2}(x_2) \triangleq f_b(x_2)$ etc.
  - The messages from internal nodes then evaluate to:

$$\begin{aligned}
\vec{\mu}_{X_3}(x_3) &= \sum_{x_1, x_2} f_a(x_1)\, f_b(x_2)\, f_c(x_1, x_2, x_3) \\
&= \sum_{x_1, x_2} \vec{\mu}_{X_1}(x_1) \vec{\mu}_{X_2}(x_2)\, f_c(x_1, x_2, x_3) \\
\overleftarrow{\mu}_{X_5}(x_5) &= \sum_{x_6, x_7} f_f(x_5, x_6, x_7)\, f_g(x_7) \\
&= \sum_{x_6, x_7} \vec{\mu}_{X_7}(x_7)\, f_f(x_5, x_6, x_7) \\
\overleftarrow{\mu}_{X_3}(x_3) &= \sum_{x_4, x_5} \overleftarrow{\mu}_{X_5}(x_5)\, f_d(x_4)\, f_e(x_3, x_4, x_5) \\
&= \sum_{x_4, x_5} \overleftarrow{\mu}_{X_5}(x_5)\, \vec{\mu}_{X_4}(x_4)\, f_e(x_3, x_4, x_5)
\end{aligned}$$

  - Crucially, all message update rules can be computed from information that is **locally available** at each node.
- This recursive pattern for computing messages applies generally and is called the **Sum-Product update rule**:

$$\vec{\mu}_Y(y) = \sum_{x_1, \ldots, x_n} \vec{\mu}_{X_1}(x_1) \cdots \vec{\mu}_{X_n}(x_n)\, f(y, x_1, \ldots, x_n)$$



- If the factor graph for a function f has **no cycles**, then the marginal $\bar{f}(x_3) = \sum_{x_1,x_2,x_4,x_5,x_6,x_7} f(x_1, x_2, \ldots, x_7)$ is given by (the **Sum-Product Theorem**)

$$\bar{f}(x_3) = \vec{\mu}_{X_3}(x_3) \cdot \overleftarrow{\mu}_{X_3}(x_3)$$

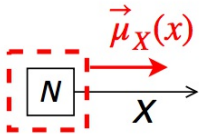
$\Rightarrow$ The marginal $\bar{f}(x_3) = \sum_{x_1,x_2,x_4,x_5,x_6,x_7} f(x_1, x_2, \ldots, x_7)$ can be efficiently computed through sum-product messages. Executing inference through SP message passing is called the **Sum-Product Algorithm**.

- Verfiy for yourself that all maginals in a cycle-free graph (a tree) can be computed by starting with messages at the terminals and working towards the root of the tree.

Half-edges and Terminal Nodes

- In order to complete the message passing framework, we need to deal with graph terminals and half-edges.
- Let us first consider a **Gaussian factor** with given parameters $m = \hat{m}$ and $V = \hat{V}$:

$$f_{\mathcal{N}}(x) = \mathcal{N}(x|\hat{m}, \hat{V}) \,.$$



- The outgoing sum-product message of this node is

$$\vec{\mu}_X(x) = f_{\mathcal{N}}(x)$$

since there are no internal variables to be integrated over (nor any incoming messages). Nodes that are connected to one edge only are called **Terminal Nodes**.

- Note that the **outgoing message for a terminal node is the factor itself**.
- Terminal nodes are very convenient to describe
  - **prior distributions** over parameters, e.g.,

$$f_\Theta(\theta) = \begin{cases} 1 & \text{for } 0 \leq \theta \leq 1 \\ 0 & \text{otherwise} \end{cases};$$

  - **initial conditions** (which are also prior distributions) for state variables, e.g.,

$$f_Z(z_0) = \mathcal{N}(\theta|1, 2)$$

    specifies out state of knowledge about Z at $t = 0$;
  - **observed variables**, e.g., use a factor

$$f_Y(y) = \delta(y - 3)$$

    to terminate the edge for variable Y if $y = 3$ is observed.
- Terminal nodes for *observed* variables interface to the rest of the graph by sending messages as delta distributions. We usually draw terminal nodes for observed variables in the graph by smaller solid-black squares. This is just to help the visualization of the graph, since the computational rules are no different than for other nodes (see cell below).
- Finally, terminal nodes can also be related to unobserved half-edges. Consider the two models

$$f(x_1, x_2) = f_a(x_1) \cdot f_b(x_1, x_2)$$
$$f'(x_1, x_2) = f_a(x_1) \cdot f_b(x_1, x_2) \cdot f_c(x_2)$$

with $f_c(x_2) = 1$. For all valid configurations, both models evaluate to exactly the same value. Hence, the **message coming from the open end of a half-edge always equals** 1 and can be understood as the outgoing message from a terminal node with an uninformative (uniform) distribution.

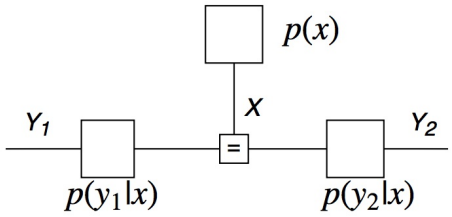
- In short, we use terminal nodes to interface the graph to the external world by setting priors on states and parameters, by specifying observations (and, mostly useful for software implementation, by terminating unobserved half-edges).

Processing Observations in a Factor Graph

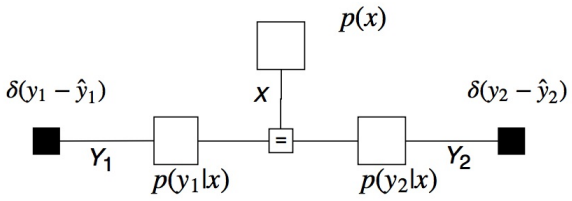
- We shortly expand here on observed nodes. Consider a generative model

$$p(x, y_1, y_2) = p(x)\, p(y_1|x)\, p(y_2|x).$$

  - This model expresses the assumption that Y_1 and Y_2 are independent measurements of X.



- Assume that we are interested in the posterior for X after observing $Y_1 = \hat{y}_1$ and $Y_2 = \hat{y}_2$. The posterior for X can be inferred by applying the sum-product algorithm to the following graph:



CODE EXAMPLE

We'll use ForneyLab, a factor graph toolbox for Julia, to build the above graph, and perform sum-product message passing to infer the posterior $p(x|y_1, y_2)$. We assume $p(y_1|x)$ and $p(y_2|x)$ to be Gaussian likelihoods with known variances:

$$p(y_1|x) = \mathcal{N}(y_1|x, v_{y1})$$
$$p(y_2|x) = \mathcal{N}(y_2|x, v_{y2})$$

Under this model, the posterior is given by:

$$\begin{aligned} p(x|y_1, y_2) &\propto \overbrace{p(y_1|x)\, p(y_2|x)}^{\text{likelihood}} \overbrace{p(x)}^{\text{prior}} \\ &= \mathcal{N}(x|\hat{y}_1, v_{y1})\, \mathcal{N}(x|\hat{y}_2, v_{y2})\, \mathcal{N}(x|m_x, v_x) \end{aligned}$$

so we can validate the answer by solving the Gaussian multiplication manually.

```
using ForneyLab # version 0.4

# Data
y1_hat = 1.0
y2_hat = 2.0

# Construct the required nodes
g      = FactorGraph()
t_y1   = TerminalNode(y1_hat; id=:y1) # y1 is observed to be y1_hat
t_y2   = TerminalNode(y2_hat; id=:y2) # y2 is observed to be y2_hat
p_x    = PriorNode(GaussianDistribution(m=0.0, V=4.0), id=:prior_x) # node for prior p(x)
lik_y1 = GaussianNode(V=1.0; id=:likelihood_y1) # likelihood p(y1|x) = N(x,1.0)
lik_y2 = GaussianNode(V=2.0; id=:likelihood_y2) # likelihood p(y2|x) = N(x,2.0)
equ    = EqualityNode()

# Create edges to connect the nodes (from left to right in the depicted graph)
Edge(t_y1, lik_y1.i[:out]) # [y1]----[p(y1|x)]
Edge(lik_y1.i[:mean], equ) # [p(y1|x)]----[=]
edge_X = Edge(p_x, equ)    # This edge represents X, and we want the sum-product msgs in both dire
ctions on this edge
Edge(equ, lik_y2.i[:mean]) # [=]----[p(y2|x)]
Edge(lik_y2.i[:out], t_y2) # [p(y1|x)]----[y2]

# Perform sum-product message passing
sum_product_algo = SumProduct(edge_X) # Automatically derives a message passing schedule
run(sum_product_algo) # Execute the message passing algorithm
x_marginal = ensureParameters!(edge_X.marginal,(:m,:V)) # make sure marginal has mean-variance par
ametrization
println("Sum-product message passing result: p(x|y1,y2) = $(x_marginal)")

# Calculate mean and variance of p(x|y1,y2) manually by multiplying 3 Gaussians (see lesson 4 for
details)
v = 1 / (1/4 + 1/1 + 1/2)
m = v * (0/4 + y1_hat/1.0 + y2_hat/2.0)
println("Manual result: p(x|y1,y2) = N(m=$(round(m,2)), V=$(round(v,2)))")
```

```
Sum-product message passing result: p(x|y1,y2) = N(m=1.14, V=0.57)
```

END OF CODE EXAMPLE

Exercise

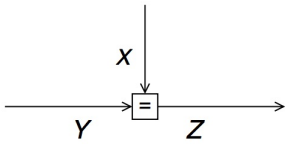
Reflect on the fact that we now have methods for both marginalization and processing observations in FFGs. In principle, we are sufficiently equipped to do inference in probabilistic models through message passing. Draw the graph for

$$p(x_1, x_2, x_3) = f_a(x_1) \cdot f_b(x_1, x_2) \cdot f_c(x_2, x_3)$$

and show which boxes need to be closed for computing $p(x_1|x_2)$.

Example: SP Messages for the Equality Node

- Let´s compute the SP messages for the **equality node** $f_=(x,y,z) = \delta(z-x)\delta(z-y)$:



$$\begin{aligned}\vec{\mu}_Z(z) &= \int \vec{\mu}_X(x)\vec{\mu}_Y(y)\,\delta(z-x)\delta(z-y)\,\mathrm{d}x\mathrm{d}y \\ &= \vec{\mu}_X(z)\int \vec{\mu}_Y(y)\,\delta(z-y)\,\mathrm{d}y \\ &= \vec{\mu}_X(z)\vec{\mu}_Y(z)\end{aligned}$$

- By symmetry, this also implies (for the same equality node) that

$$\begin{aligned}\overleftarrow{\mu}_X(x) &= \vec{\mu}_Y(x)\overleftarrow{\mu}_Z(x) \quad \text{and} \\ \overleftarrow{\mu}_Y(y) &= \vec{\mu}_X(y)\overleftarrow{\mu}_Z(y)\,.\end{aligned}$$

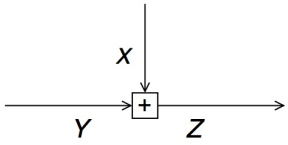
- Let us now consider the case of Gaussian messages $\vec{\mu}_X(x) = \mathcal{N}(\vec{m}_X, \vec{V}_X)$, $\vec{\mu}_Y(y) = \mathcal{N}(\vec{m}_Y, \vec{V}_Y)$ and $\vec{\mu}_Z(z) = \mathcal{N}(\vec{m}_Z, \vec{V}_Z)$. Let´s also define the precision matrices $\vec{W}_X \triangleq \vec{V}_X^{-1}$ and similarly for Y and Z. Then applying the SP update rule leads to multiplication of two Gaussian distributions, resulting in

$$\begin{aligned}\vec{W}_Z &= \vec{W}_X + \vec{W}_Y \\ \vec{W}_Z\vec{m}_z &= \vec{W}_X\vec{m}_X + \vec{W}_Y\vec{m}_Y\end{aligned}$$

- It follows that **message passing through an equality node is similar to applying Bayes rule**, i.e., fusion of two information sources. Does this make sense?

Example: SP Messages for the Addition and Multiplication Nodes

- Next, consider an **addition node** $f_+(x,y,z) = \delta(z-x-y)$:



$$\begin{aligned}\vec{\mu}_Z(z) &= \int \vec{\mu}_X(x)\vec{\mu}_Y(y)\,\delta(z-x-y)\,\mathrm{d}x\mathrm{d}y \\ &= \int \vec{\mu}_X(z)\vec{\mu}_Y(z-x)\,\mathrm{d}x\,,\end{aligned}$$

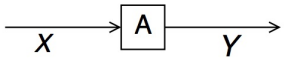
  i.e., $\vec{\mu}_Z$ is the convolution of the messages $\vec{\mu}_X$ and $\vec{\mu}_Y$.

- Of course, for Gaussian messages, these update rules evaluate to

$$\begin{aligned}\vec{m}_Z &= \vec{m}_X + \vec{m}_Y \\ \vec{V}_z &= \vec{V}_X + \vec{V}_Y\,.\end{aligned}$$

- **Exercise**: For the same summation node, work out the SP update rule for the *backward* message $\overleftarrow{\mu}_X(x)$ as a function of $\vec{\mu}_Y(y)$ and $\overleftarrow{\mu}_Z(z)$? And further refine the answer for Gaussian messages.

- Next, let us consider a **multiplication** by a fixed (invertable matrix) gain $f_A(x,y) = \delta(y-Ax)$



$$\begin{aligned}\vec{\mu}_Y(y) &= \int \vec{\mu}_X(x)\,\delta(y-Ax)\,\mathrm{d}x \\ &= \vec{\mu}_X(A^{-1}y)\,.\end{aligned}$$

- For a Gaussian message input message $\vec{\mu}_X(x) = \mathcal{N}(\vec{m}_X, \vec{V}_X)$, the output message is also Gaussian with

$$\begin{aligned}\vec{m}_Y &= A\vec{m}_X \\ \vec{V}_Y &= A\vec{V}_X A^T\end{aligned}$$

  since

$$\begin{aligned}\vec{\mu}_Y(y) &= \vec{\mu}_X(A^{-1}y) \\ &\propto \exp\left(-\frac{1}{2}\left(A^{-1}y-\vec{m}_X\right)^T\vec{V}_X^{-1}\left(A^{-1}y-\vec{m}_X\right)\right) \\ &= \exp\left(-\frac{1}{2}(y-A\vec{m}_X)^T A^{-T}\vec{V}_X^{-1}A(y-A\vec{m}_X)\right) \\ &\propto \mathcal{N}(A\vec{m}_X, A\vec{V}_X A^T)\,.\end{aligned}$$

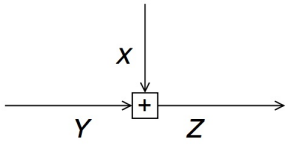
- **Excercise**: Proof that, for the same factor $\delta(y-Ax)$ and Gaussian messages, the (backward) sum-product message $\overleftarrow{\mu}_X$ is given by

$$\begin{aligned}\overleftarrow{\xi}_X &= A^T\overleftarrow{\xi}_Y \\ \overleftarrow{W}_X &= A^T\overleftarrow{W}_Y A\end{aligned}$$

  where $\overleftarrow{\xi}_X \triangleq \overleftarrow{W}_X\overleftarrow{m}_X$ and $\overleftarrow{W}_X \triangleq \overleftarrow{V}_X^{-1}$ (and similarly for Y).

CODE EXAMPLE

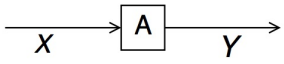
Let's calculate the Gaussian forward and backward messages for the addition node in ForneyLab.



```
# Build factor graph
g = FactorGraph()
t_X = TerminalNode(GaussianDistribution(m=1.0, V=1.0))
t_Y = TerminalNode(GaussianDistribution(m=2.0, V=1.0))
t_Z = TerminalNode(GaussianDistribution(m=4.0, V=2.0))
add_node = AdditionNode()
Edge(t_X, add_node.i[:in1])
Edge(t_Y, add_node.i[:in2])
Edge(add_node.i[:out], t_Z)

msg_forward_Z = run(SumProduct(add_node.i[:out]))
print("Forward message on Z: $(msg_forward_Z)")

msg_backward_X = run(SumProduct(add_node.i[:in1]))
print("Backward message on X: $(msg_backward_X)")
```

```
Manual result: p(x|y1,y2) = N(m=1.14, V=0.57)
Forward message on Z: ForneyLab.Message{ForneyLab.GaussianDistribution} with payloa
d N(m=3.00, V=2.00)

Backward message on X: ForneyLab.Message{ForneyLab.GaussianDistribution} with paylo
ad N(m=2.00, V=3.00)
```

In the same way we can also investigate the forward and backward messages for the gain node




```
# Build factor graph
g = FactorGraph()
t_X = TerminalNode(GaussianDistribution(m=1.0, V=1.0))
t_Y = TerminalNode(GaussianDistribution(m=2.0, V=1.0))
gain_node = GainNode(gain=4.0)
Edge(t_X, gain_node.i[:in])
Edge(gain_node.i[:out], t_Y)

msg_forward_Y = run(SumProduct(gain_node.i[:out]))
print("Forward message on Y: $(msg_forward_Y)")

msg_backward_X = run(SumProduct(gain_node.i[:in]))
# Make sure the message has mean and variance parameters instead of precision
ensureParameters!(msg_backward_X.payload, (:m,:V))
print("Backward message on X: $(msg_backward_X)")
```

```
Forward message on Y: ForneyLab.Message{ForneyLab.GaussianDistribution} with payloa
d N(m=4.00, V=16.00)

Backward message on X: ForneyLab.Message{ForneyLab.GaussianDistribution} with paylo
ad N(m=0.50, V=0.06)
```

- **Excercise**: Confirm for both examples the results of the sum-product rule by manual computation.

END OF CODE EXAMPLE

Inference in Linear Gaussian Models by Sum-Product Message Passing

- The foregoing message update rules can be extended to all scenarios involving additions, fixed-gain multiplications and branching (equality nodes), thus creating a completely **automatable inference framework** for factorized linear Gaussian models.
- The update rules for elementary and important node types can be put in a Table (see **Tables 1 through 6** in Loeliger, 2007).
- If the update rules for all node types in a graph have been tabulated, then inference by message passing comes down to a set of table-lookup operations. This also works for large graphs (where 'manual' inference becomes intractable).
- If the graph contains no cycles, the Sum-Product Algorithm computes **exact** marginals for all hidden variables.
- If the graph contains cycles, we have in principle an infinite tree without terminals. In this case, the SP Algorithm is not guaranteed to find exact marginals. In practice, if we apply the SP algorithm for just a few iterations we often find satisfying approximate marginals.

Example

Consider a variable X with measurements $D = \{x_1, x_2\}$. We assume the following model for X:

$$p(D, \theta) = p(\theta) \cdot \prod_{n=1}^{2} p(x_n|\theta)$$

$$p(\theta) = \mathcal{N}(\theta \mid 0, 1)$$

$$p(x_n \mid \theta) = \mathcal{N}(x_n \mid \theta, 1)$$

Draw the factor graph and infer θ through the Sum-Product Algorithm.

Dynamic Latent Variable Models

Preliminaries

- Goal
  - Introduction to dynamic (=temporal) Latent Variable Models, including the Hidden Markov Model and Kalman filter.
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp.605-615 on Hidden Markov Models
    - Bishop pp.635-641 on Kalman filters
    - Faragher (2012), Understanding the Basis of the Kalman Filter, and video
    - Minka (1999), From Hidden Markov Models to Linear Dynamical Systems

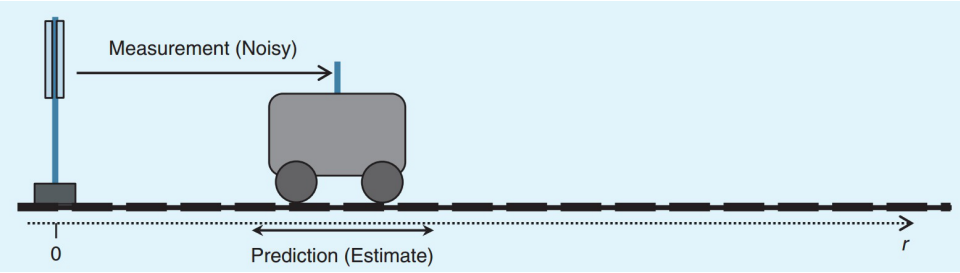
Example Problem

- We consider a one-dimensional cart position tracking problem, see Faragher 2012.
- The hidden states are the position z_t and velocity $\dot{z}_t$. We can apply an external acceleration/breaking force u_t. (Noisy) observations are represented by x_t.
- The equations of motions are given by

$$\begin{bmatrix} z_t \\ \dot{z}_t \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} z_{t-1} \\ \dot{z}_{t-1} \end{bmatrix} + \begin{bmatrix} (\Delta t)^2/2 \\ \Delta t \end{bmatrix} u_t + \mathcal{N}(0, \Sigma_z)$$

$$x_t = \begin{bmatrix} z_t \\ \dot{z}_t \end{bmatrix} + \mathcal{N}(0, \Sigma_x)$$

- Infer the position after 10 time steps.

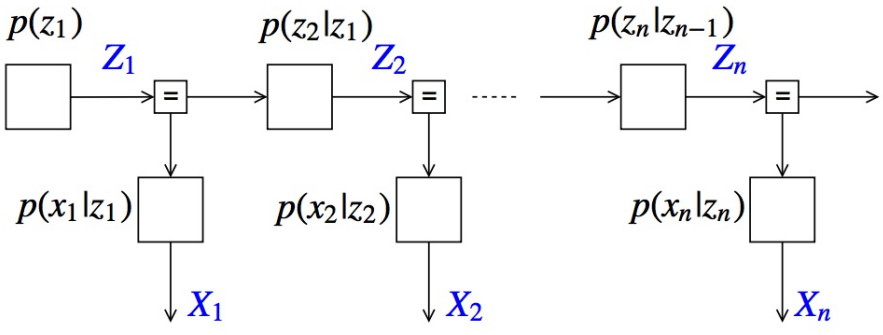


Dynamical Models

- In this lesson, we consider models where the sequence order of observations matters.
- Consider the *ordered* observation sequence
$$x^N \triangleq (x_1, x_2, \ldots, x_N) \,.$$
- We wish to develop a generative model
$$p(x^N \,|\, \theta)$$
that 'explains' the time series x^N.
- We cannot use the IID assumption $p(x^N|\theta) = \prod_n p(x_n \,|\, \theta)$. In general, we *can* use the **chain rule** (a.k.a. the general product rule)
$$\begin{aligned} p(x^N) &= p(x_N|x^{N-1})\, p(x^{N-1}) \\ &= p(x_N|x^{N-1})\, p(x_{N-1}|x^{N-2}) \cdots p(x_2|x_1)\, p(x_1) \\ &= p(x_1) \prod_{n=2}^{N} p(x_n \,|\, x^{n-1}) \end{aligned}$$
- Generally, we will want to limit the depth of dependencies on previous observations. For example, the Mth-order linear **Auto-Regressive** (AR) model
$$p(x_n|x^{n-1}) = \mathcal{N}\left(\sum_{m=1}^{M} a_m x_{n-m} \,, \sigma^2 \right)$$
limits the dependencies to the past M samples.

State-space Models

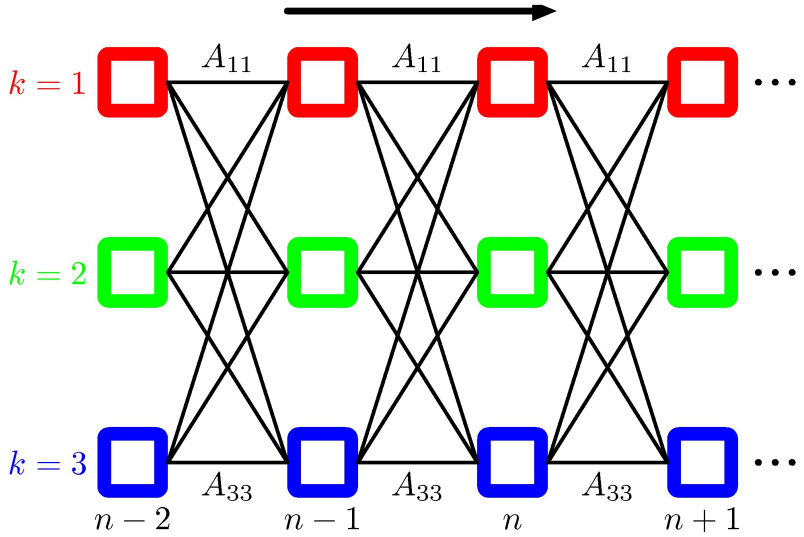
- A limitation of AR models is that they need a lot of parameters in order to create a flexible model. E.g., if x_n is an K-dimensional discrete variable, then an Mth-order AR model will have $K^{M-1}(K-1)$ parameters.
- Similarly to Gaussian Mixture models and latent Factor models, we can create a flexible dynamic system by introducing *latent* (unobserved) variables $z^N \triangleq (z_1, z_2, \ldots, z_N)$ (one z_n for each observation x_n). In dynamic systems, the latent variables z_n are usually called *state variables*.
- A general **state space model** is defined by
$$\begin{aligned} \text{a state transition model: } & p(z_n \,|\, z^{n-1}) \,, \\ \text{an observation model: } & p(x_n \,|\, z_n) \,, \\ \text{and an initial state: } & p(z_1) \end{aligned}$$
- A very common computational assumption is to let state transitions be ruled by a *first-order Markov chain* as
$$p(z_n \,|\, z^{n-1}) = p(z_n \,|\, z_{n-1})$$
- The Markov assumption leads to the following joint probability distribution for the state-space model:
$$p(x^N, z^N) = \underbrace{p(z_1)}_{\text{initial state}} \prod_{n=2}^{N} \underbrace{p(z_n \,|\, z_{n-1})}_{\text{state transitions}} \prod_{n=1}^{N} \underbrace{p(x_n \,|\, z_n)}_{\text{observations}}$$
- The Forney-style factor graph for a state-space model:



- Exercise: Show that in a state-space model x_n is not a first-order Markov chain in the observations, i.e., show that
$$p(x_n \,|\, x_{n-1}, x_{n-2}) \neq p(x_n \,|\, x_{n-1}) \,.$$

Hidden Markov Models, Kalman filters etc.

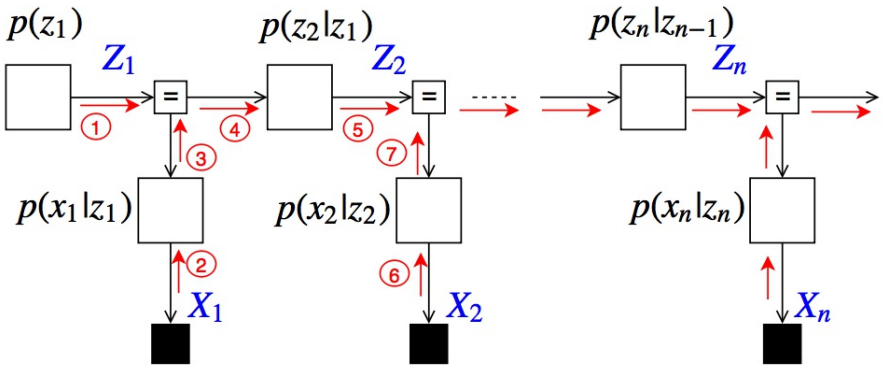
- A **Hidden Markov Model** (HMM) is a state-space model with discrete-valued state variables Z_n.
  - E.g., Z_n is a K -dimensional hidden binary 'class indicator' with transition probabilities $A_{jk} \triangleq p(z_{nk} = 1 \mid z_{n-1,j} = 1)$, or equivalently
    $$p(z_n|z_{n-1}) = \prod_{k=1}^{K}\prod_{j=1}^{K} A_{jk}^{z_{n-1,j}\, z_{nk}}$$
    which is usually accompanied by an initial state distribution $\pi_k \triangleq p(z_{1k} = 1)$.
  - The classical HMM has also discrete-valued observations but in pratice any (probabilistic) observation model $p(x_n|z_n)$ may be coupled to the hidden Markov chain.



- Another well-known state-space model with continuous-valued state variables Z_n is the **(Linear) Dynamical System** (LDS), which is defined as
  $$\begin{aligned} p(z_n \mid z_{n-1}) &= \mathcal{N}(\, Az_{n-1}\,,\, \Sigma_z\,) \\ p(x_n \mid z_n) &= \mathcal{N}(\, Cz_n\,,\, \Sigma_x\,) \\ p(z_1) &= \mathcal{N}(\, \mu_1\,,\, \Sigma_1\,) \end{aligned}$$
- Note that the joint distribution over $\{x_1, z_1, \ldots, x_n, z_n\}$ is a (large-dimensional) Gaussian distribution. This means that, in principle, every inference problem on the generative model also leads to a Gaussian distribution.
- Technically, a **Kalman filter** is the solution to the recursive estimation (=inference) of the hidden state z_n based on past observations in an LDS, i.e., Kalman filtering solves the problem $p(z_n \mid x^n)$ based on the previous estimate $p(z_{n-1} \mid x^{n-1})$ and a new observation x_n (given the generative model specification).
- Kalman filtering and hidden Markov models (and variants thereof) are at the basis of a wide range of complex information processing systems, such as speech and language recognition, robotics and automatic car navigation, and even processing of DNA sequences.

Message Passing in State-space Models

- Once the (state-space) models have been specified, we can define state and parameter estimation problems as inference tasks on the generative model.
- In principle, for linear Gaussian models these inference tasks can be analytically solved, see e.g. Faragher, 2012
  - These derivations quickly become quite laborious
- Alternatively, we could specify the generative model in a (Forney-style) factor graph and use automated message passing to infer the posterior over the hidden variables. E.g., the message passing schedule for Kalman filtering looks like this:



Example Problem Revisited

We can solve the cart tracking problem by sum-product message passing in a factor graph like the one depicted above. All we have to do is create factor nodes for the state-transition model $p(z_t|z_{t-1})$ and the observation model $p(x_t|z_t)$. Then we just build the factor graph and let ForneyLab (factor graph toolbox) perform message passing.

We'll implement the following model:

$$\begin{bmatrix} z_t \\ \dot{z}_t \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} z_{t-1} \\ \dot{z}_{t-1} \end{bmatrix} + \begin{bmatrix} (\Delta t)^2/2 \\ \Delta t \end{bmatrix} u_t + \mathcal{N}(0, \Sigma_z)$$

$$\mathbf{x}_t = \begin{bmatrix} z_t \\ \dot{z}_t \end{bmatrix} + \mathcal{N}(0, \Sigma_x)$$

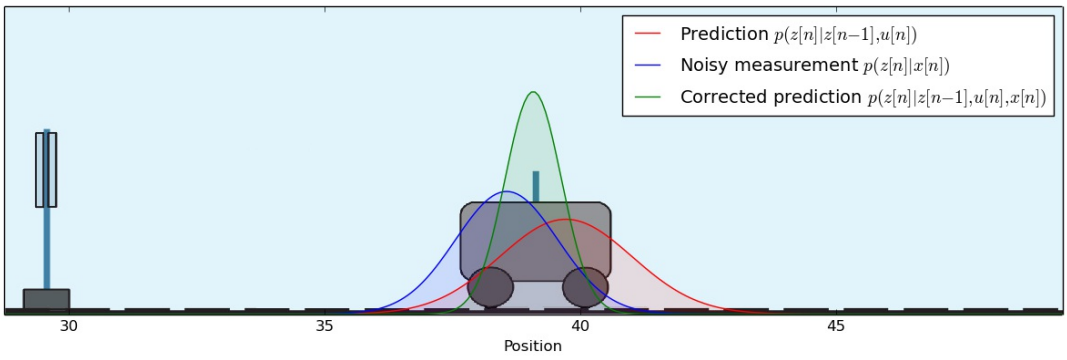
```
include("scripts/cart_tracking_helpers.jl") # implements required factor nodes + helper functions

# Specify the model parameters
Δt = 1.0                    # assume the time steps to be equal in size
A = [1.0 Δt;
     0.0 1.0]
b = [0.5*Δt^2; Δt]
Σz = diagm([0.2*Δt; 0.1*Δt]) # process noise covariance
Σx = diagm([1.0; 2.0])       # observation noise covariance;

# Generate noisy observations
n = 10                # perform 10 timesteps
z_start = [10.0; 2.0] # initial state
u = 0.2 * ones(n)     # constant input u
noisy_x = generateNoisyMeasurements(z_start, u, A, b, Σz, Σx);
```

Since the factor graph is just a concatination of n identical "sections", we only have to specify a single section, and ForneyLab will automatically build a 'virtual' factor graph consisting of n sections. Let's build a section of the factor graph:

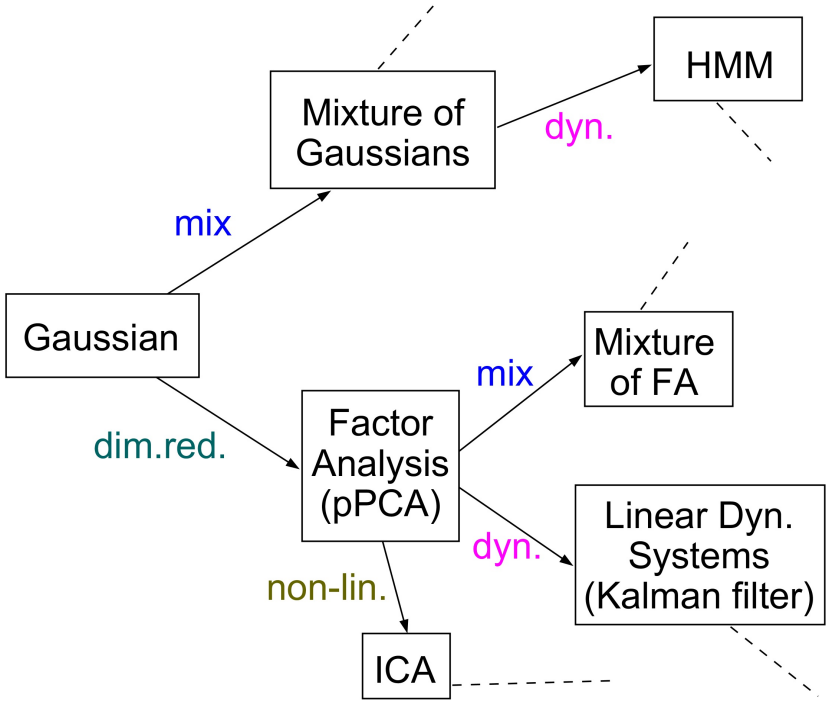
```
fg = FactorGraph()
n_z_old      = TerminalNode(vague(MvGaussianDistribution{2}), id=:z_old)       # terminal for z[t-1]
n_z_new      = TerminalNode(vague(MvGaussianDistribution{2}), id=:z_new)       # terminal for z[t]
n_u          = TerminalNode(vague(GaussianDistribution), id=:u)                # terminal for u[t]
n_x          = TerminalNode(MvDeltaDistribution(zeros(2)), id=:x)              # terminal for u[t]
n_transition = LinearTransitionModelNode(A=A, b=b, Σ=Σz, id=:transition_model) # node p(z_new|z_old,u;A,b,Σ)
n_equality   = EqualityNode()
n_obs        = GaussianNode(V=Σx, id=:observation_model)                       # node p(x|z)

Edge(n_z_old, n_transition.i[:z_old])
Edge(n_u, n_transition.i[:u])
Edge(n_transition.i[:z_new], n_equality.i[1])
Edge(n_equality.i[2], n_obs.i[:mean])
Edge(n_obs.i[:out], n_x)
Edge(n_equality.i[3], n_z_new)

Wrap(n_z_new, n_z_old) # Tell ForneyLab how to concatinate the sections
# draw(fg);
```

Now that we've built the factor graph, we can perform Kalman filtering by inserting measurement data into the factor graph and performing message passing.

```
# Insert data into graph, perform message-passing
attachReadBuffer(n_u, map(GaussianDistribution, u))      # insert values of u into terminal node n_u
attachReadBuffer(n_x, map(MvDeltaDistribution, noisy_x)) # insert the noisy measurements into terminal node n_x
run(SumProduct()) # Perform sum-product message passing

# Collect prediction p(z[n]|z[n-1]), measurement p(z[n]|x[n]), corrected prediction p(z[n]|z[n-1],x[n])
prediction      = n_transition.i[:z_new].message.payload
measurement     = n_obs.i[:mean].message.payload
corr_prediction = n_equality.i[3].message.payload

# Make a fancy plot of the prediction, noisy measurement, and corrected prediction after n timesteps
plotCartPrediction(prediction, measurement, corr_prediction);
```



Extensions


HMM
Mixture of Gaussians
dyn.
mix
Gaussian
Mixture of FA
mix
dim.red.
Factor Analysis (pPCA)
Linear Dyn. Systems (Kalman filter)
dyn.
non-lin.
ICA

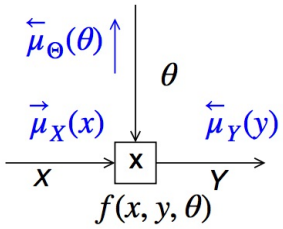
EM as a Message Passing Algorithm

Preliminaries

- Goals
  - Describe Expectation-Maximization (EM) as a message passing algorithm on a Forney-style factor graph
- Materials
  - Madatory
    - These lecture notes
  - Optional
    - Dauwels et al., 2009, pp. 1-5 (sections I,II and III)

A Problem for the Multiplier Node

- Consider the multiplier factor $f(x,y,\theta) = \delta(y-\theta x)$ with incoming Gaussian messages $\vec{\mu}_X(x) = \mathcal{N}(x|m_x, v_x)$ and $\overleftarrow{\mu}_Y(y) = \mathcal{N}(y|m_y, v_y)$. For simplicity's sake, we assume all variables are scalar.



- In a system identification setting, we are interested in computing the outgoing message $\overleftarrow{\mu}_\Theta(\theta)$.
- Let's compute the sum-product message:

$$\begin{aligned}
\overleftarrow{\mu}_\Theta(\theta) &= \int \vec{\mu}_X(x)\, \overleftarrow{\mu}_Y(y)\, f(x,y,\theta)\, \mathrm{d}x\mathrm{d}y \\
&= \int \mathcal{N}(x \,|\, m_x, v_x)\, \mathcal{N}(y \,|\, m_y, v_y)\, \delta(y-\theta x)\, \mathrm{d}x\mathrm{d}y \\
&= \int \mathcal{N}(x \,|\, m_x, v_x)\, \mathcal{N}(\theta x \,|\, m_y, v_y)\, \mathrm{d}x \\
&= \int \mathcal{N}(x \,|\, m_x, v_x)\, \mathcal{N}\left(x \,\middle|\, \frac{m_y}{\theta}, \frac{v_y}{\theta^2}\right) \mathrm{d}x \\
&= \mathcal{N}\left(\frac{m_y}{\theta} \,\middle|\, m_x, v_x + \frac{v_y}{\theta^2}\right) \cdot \int \mathcal{N}(x \,|\, m_*, v_*)\, \mathrm{d}x \qquad \text{(SRG-6)} \\
&= \mathcal{N}\left(\frac{m_y}{\theta} \,\middle|\, m_x, v_x + \frac{v_y}{\theta^2}\right)
\end{aligned}$$

- This is **not** a Gaussian message for Θ! Passing this message into the graph leads to very serious problems when trying to compute sum-product messages for other factors in the graph.
  - (We have seen before in the lesson on Working with Gaussians that multiplication of two Gaussian-distributed variables does *not* produce a Gaussian distributed variable.)
- The same problem occurs in a forward message passing schedule when we try to compute a message for Y from incoming Gaussian messages for both X and Θ.

Limitations of Sum-Product Messages

- The foregoing example shows that the sum-product (SP) message update rule will sometimes not do the job. For example:
  - On large-dimensional **discrete** domains, the SP update rule maybe computationally intractable.
  - On **continuous** domains, the SP update rule may not have a closed-form solution or the rule may lead to a function that is incompatible with Gaussian message passing.
- There are various ways to cope with 'intractable' SP update rules. In this lesson, we discuss how the EM-algorithm can be written as a message passing algorithm on factor graphs. Then, we will solve the 'multiplier node problem' with EM messages (rather than with SP messages).

EM as Message Passing

- Consider first a general setting with likelihood function $f(x,\theta)$, hidden variables x and tuning parameters θ. Assume that we are interested in the maximum likelihood estimate

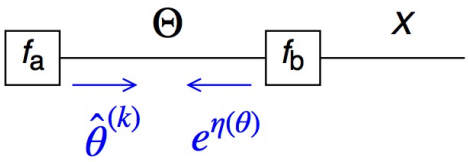
$$\hat{\theta} = \arg\max_{\theta} \int f(x,\theta) \mathrm{d}x \,.$$

- If $\int f(x,\theta)\mathrm{d}x$ is intractible, we can try to apply the EM-algorithm to estimate $\hat{\theta}$, which leads to the following iterations (*cf.* lesson on the EM algorithm):

$$\hat{\theta}^{(k+1)} = \underbrace{\arg\max_{\theta}}_{\text{M-step}} \left(\underbrace{\int_x f(x,\hat{\theta}^{(k)}) \, \log f(x,\theta) \, \mathrm{d}x}_{\text{E-step}} \right)$$

- It turns out that *for factorized functions* $f(x,\theta)$, the EM-algorihm can be executed as a message passing algorithm on the factor graph.
- As an simple example, we consider the factorization

$$f(x,\theta) = f_a(\theta) f_b(x,\theta)$$



- Applying the EM-algorithm to this graph leads to the following forward and backward messages over the θ edge

$$\textbf{E-step}: \quad \eta(\theta) = \int p_b(x|\hat{\theta}^{(k)}) \log f_b(x,\theta) \, \mathrm{d}x$$

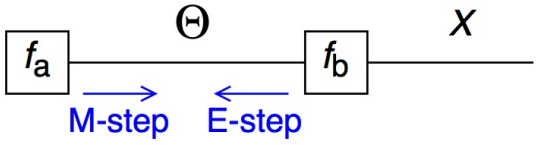
$$\textbf{M-step}: \quad \hat{\theta}^{(k+1)} = \arg\max_{\theta} \left(f_a(\theta) \, e^{\eta(\theta)} \right)$$

where $p_b(x|\hat{\theta}^{(k)}) \triangleq \frac{f_b(x,\hat{\theta}^{(k)})}{\int f_b(x',\hat{\theta}^{(k)}) \, \mathrm{d}x'}$.

Proof:

$$
\begin{aligned}
\hat{\theta}^{(k+1)} &= \arg\max_{\theta} \int_x f(x, \hat{\theta}^{(k)}) \, \log f(x, \theta) \, \mathrm{d}x \\
&= \arg\max_{\theta} \int_x f_a(\theta) f_b(x, \hat{\theta}^{(k)}) \, \log(f_a(\theta) \, f_b(x, \theta)) \, \mathrm{d}x \\
&= \arg\max_{\theta} \int_x f_b(x, \hat{\theta}^{(k)}) \cdot (\log f_a(\theta) + \log f_b(x, \theta)) \, \mathrm{d}x \\
&= \arg\max_{\theta} \left(\log f_a(\theta) + \frac{\int f_b(x, \hat{\theta}^{(k)}) \log f_b(x, \theta) \, \mathrm{d}x}{\int f_b(x', \hat{\theta}^{(k)}) \, \mathrm{d}x'} \right) \\
&= \arg\max_{\theta} \left(\log f_a(\theta) + \underbrace{\int p_b(x|\hat{\theta}^{(k)}) \log f_b(x, \theta) \, \mathrm{d}x}_{\eta(\theta)} \right) \\
&= \underbrace{\arg\max_{\theta}}_{\text{M-step}} \left(f_a(\theta) \underbrace{e^{\eta(\theta)}}_{\text{E-step}} \right)
\end{aligned}
$$

- The messages represent the 'E' and 'M' steps, respectively:



- The quantity $\eta(\theta)$ (a.k.a. the **E-log message**) may be interpreted as a log-domain summary of f_b. The message $e^{\eta(\theta)}$ is the corresponding 'probability domain' message that is consistent with the semantics of messages as summaries of factors. In a software implementation, you can use either domain, as long as a consistent method is chosen.
- Note that the denominator $\int f_b(x', \hat{\theta}^{(k)}) \, \mathrm{d}x'$ in p_b is just a scaling factor that can usually be ignored, leading to a simpler E-log message

$$\eta(\theta) = \int f_b(x, \hat{\theta}^{(k)}) \log f_b(x, \theta) \, \mathrm{d}x \, .$$

EM vs SP and MP Message Passing

- Consifer again the likelihood model $f(x,\theta)$ with x a set of hidden variables. We are interested in the ML estimate

$$\hat{\theta} = \arg\max_{\theta} \int f(x,\theta)\mathrm{d}x\,.$$

- Recall that in a 'regular' (*not* message passing) setting, the EM-algorithm is particularly useful when the *expectation* (E-step)

$$\eta(\theta) = \int_x f(x,\hat{\theta}^{(k)})\,\log f(x,\theta)\,\mathrm{d}x$$

leads to easier expressions than the *marginalization* (which is what we really want)

$$\bar{f}\,(\theta) = \int f(x,\theta)\mathrm{d}x.$$

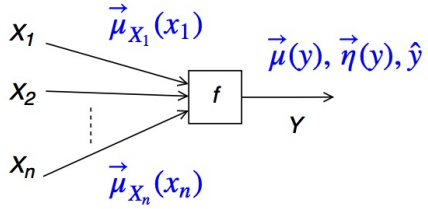
- Similarly, in a message passing framework with connected nodes f_a and f_b, EM messages are particularly useful when the *expectation* (represented by the **E-log message**)

$$\eta(\theta) = \int f_b(x|\hat{\theta}^{(k)})\log f_b(x,\theta)\,\mathrm{d}x$$

leads to easier expressions than the *marginalization* (represented by the **sum-product message**, which is also what we really want)

$$\mu(\theta) = \int f_b(x,\theta)\mathrm{d}x.$$

- Just as for the sum-product (SP) and max-product (MP) messages, we can work out the outgoing E-log message on the Y edge for a *general* node $f(x_1,\ldots,x_M,y)$ with given message inputs $\vec{\mu}_{X_m}(x_m)$ (see also Dauwels et al. (2009), Table-1, pg.4):



$$\textbf{SP}:\ \vec{\mu}(y) = \int \vec{\mu}_{X_1}(x_1)\cdots\vec{\mu}_{X_M}(x_M)\,f(x_1,\ldots,x_M,y)\,\mathrm{d}x_1\ldots\mathrm{d}x_M$$

$$\textbf{MP}:\ \hat{y} = \arg\max_{x_1,\ldots,x_M} \vec{\mu}_{X_1}(x_1)\cdots\vec{\mu}_{X_M}(x_M)\,f(x_1,\ldots,x_M,y)$$

$$\textbf{E-log}:\ \vec{\eta}(y) = \int p(x_1,\ldots,x_M|y^{(k)})\,\log f(x_1,\ldots,x_M,y)\,\mathrm{d}x_1\ldots\mathrm{d}x_M$$

where $p(x_1,\ldots,x_M|y^{(k)}) \triangleq \frac{\vec{\mu}_{X_1}(x_1)\cdots\vec{\mu}_{X_M}(x_M)\,f(x_1,\ldots,x_M,\hat{y}^{(k)})}{\int\vec{\mu}_{X_1}(x_1)\cdots\vec{\mu}_{X_M}(x_M)\,f(x_1,\ldots,x_M,\hat{y}^{(k)})\,\mathrm{d}x_1\ldots\mathrm{d}x_M}$.

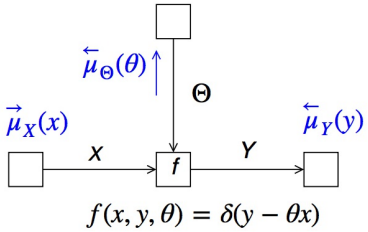
- **Exercise**: proof the generic E-log message update rule.

A Snag for EM Message Passing on Deterministic Nodes

- The factors for deterministic nodes are (Dirac) delta functions, e.g., $\delta(y-\theta x)$ for the multiplier.
- Note that the outgoing E-log message for a deterministic node will also be a delta function, since the expectation of $\log\delta(\cdot)$ is again a delta function. For details, consult Dauwels et al. (2009) pg.5, section F.
- This would stall the iterative estimation process at the current estimate since the outgoing E-log message would express complete certainty about the estimate.
- This issue can be resolved by closing a box around a subgraph that includes (the deterministic node) f and *at least one non-deterministic factor*. EM message passing can now proceed with the newly created node.

A Solution for the Multiplier Node with Unknown Coefficient

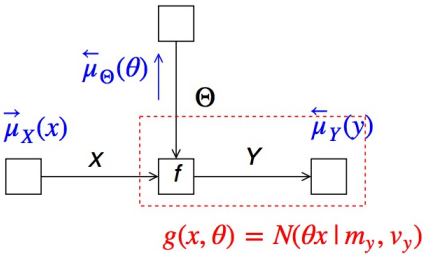
- We get back no to the original problem in this lesson. Consider again the (scalar) multiplier with unknown coefficient $f(x,y,\theta) = \delta(y-\theta x)$ and incoming messages $\overrightarrow{\mu_X}(x) = \mathcal{N}(x|m_x,v_x)$ and $\overleftarrow{\mu_Y}(y) = \mathcal{N}(y|m_y,v_y)$.



  We will now compute the outgoing E-log message for Θ.

- Since $f(x,y,\theta)$ is deterministic, we will first group f with the (non-deterministic) node $\overleftarrow{\mu_Y}(y) = \mathcal{N}(y|m_y,v_y)$, leading (through sum-product rule) to

$$\begin{aligned} g(x,\theta) &\triangleq \int \overleftarrow{\mu_Y}(y)\, f(x,y,\theta)\, \mathrm{d}y \\ &= \int \mathcal{N}(y|m_y,v_y)\, \delta(y-\theta x)\, \mathrm{d}y \\ &= \mathcal{N}(\theta x \mid m_y, v_y)\,. \end{aligned}$$



- The problem now is to pass an E-log message out of $g(x,\theta)$. Assume that g has received an estimate $\hat{\theta}$ from the incoming message over the Θ edge. The E-log update rule then prescribes

$$\begin{aligned} \eta(\theta) &= \mathbb{E}\left[\log g(x,\theta)\right] \\ &= \mathbb{E}\left[\mathcal{N}(\theta x|m_y,v_y)\right] \\ &= \text{const.} - \frac{1}{2v_y}\left(\mathbb{E}[X^2]\theta^2 - 2m_y\mathbb{E}[X]\theta + m_y^2\right) \\ &\propto \mathcal{N}_\xi\left(\theta \,\middle|\, \frac{m_y\mathbb{E}[X]}{v_y}, \frac{\mathbb{E}\left[X^2\right]}{v_y}\right) \end{aligned}$$

  where we used the 'canonical' parametrization of the Gaussian $\mathcal{N}_\xi(\theta \mid \xi, w) \propto \exp\left(\xi\theta - \frac{1}{2}w\theta^2\right)$.

- In the E-log message update rule, the expections $\mathbb{E}\left[X\right]$ and $\mathbb{E}\left[X^2\right]$ have to be taken w.r.t. $p(x|\hat{\theta}) = \overrightarrow{\mu_X}(x)\, g(x,\hat{\theta})$ (consult the generic E-log update rule). A straightforward (but rather painful) derivation leads to

$$\begin{aligned} p(x \mid \hat{\theta}) &= \overrightarrow{\mu_X}(x)\, g(x,\hat{\theta}) \\ &= \mathcal{N}(x \mid m_x, v_x) \cdot \mathcal{N}(\hat{\theta}x \mid m_y, v_y) \\ &= \mathcal{N}(x \mid m_x, v_x) \cdot \mathcal{N}\left(x \,\middle|\, \frac{m_y}{\hat{\theta}}, \frac{v_y}{\hat{\theta}^2}\right) \\ &\propto \mathcal{N}_\xi(x \mid \xi_g, w_g) \end{aligned}$$

  where $w_g = \frac{1}{v_x} + \frac{\hat{\theta}^2}{v_y}$ and $\xi_g \triangleq w_g m_g = \frac{m_x}{v_x} + \frac{\hat{\theta}\, m_y}{v_y}$. It follows that

$$\begin{aligned} \mathbb{E}\left[X\right] &= m_g \\ \mathbb{E}\left[X^2\right] &= m_g^2 + w_g^{-1} \end{aligned}$$

- $\Rightarrow$ The E-log update formula may not be fun to derive, but the result is very pleasing: the **E-log message for the multiplier with unknown coefficient is a Gaussian message** with closed-form expressions for its parameters! See also Dauwels et al. (2009) Table-2, pg.6.

Automating Inference

- It follows that, for a dynamical system with unknown coefficients, both state estimation and parameter learning can be achieved through Gaussian message passing based on SP and EM message update rules.
- These (SP and EM) message update rules can be tabularized and implemented in software for a large set of factors that are common in probabilistic models. (See the tables in Loeliger et al. (2007) and Dauwels et al. (2009)).
- Tabulated SP and EM messages for frequently occuring factors facilitate the automated derivation of nontrivial inference algorithms.
- This makes it possible to automate inference for state and parameter estimation in very complex probabilistic model. Here (in the SPS group at TU/e), we are developing such a factor graph toolbox in Julia.
- There is lots more to say about factor graphs. This is a very exciting area of research that promises both
    1. to consolidate a wide range of signal processing and machine learning algorithms in one elegant framework
    2. to automate inference and learning in new models that have previously been untractable for existing machine learning methods.

Example: Linear Dynamical Systems

As before let us consider the linear dynamical system (LDS)

$$\begin{aligned} z_n &= A z_{n-1} + w_n \\ x_n &= C z_n + v_n \\ w_n &\sim \mathcal{N}(0, \Sigma_w) \\ v_n &\sim \mathcal{N}(0, \Sigma_v) \end{aligned}$$

Again, we will consider the case where x_n is observed and z_n is a hidden state. C, Σ_w and Σ_v are given parameters but in contrast to the previous section, we will assume that the value of parameter A is unknown.